UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

CASOS ESPECIAIS ÓTIMOS DE ALGORITMOS APROXIMATIVOS PARA PROBLEMAS DE ESCALONAMENTO COM RESTRIÇÕES DE PRECEDÊNCIA EM PROCESSADORES PARALELOS IDÊNTICOS

ELTON CARLOS COSTA LEVER

Março de 2017

Manaus - AM

ELTON CARLOS COSTA LEVER

CASOS ESPECIAIS ÓTIMOS DE ALGORITMOS APROXIMATIVOS PARA PROBLEMAS DE ESCALONAMENTO COM RESTRIÇÕES DE PRECEDÊNCIA EM PROCESSADORES PARALELOS IDÊNTICOS

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas (PPGI/IComp, UFAM) como parte dos requisitos necessários à obtenção do título de Mestre em Informática

Orientadora: Rosiane de Freitas , D.Sc.

Março de 2017

Manaus - AM

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Lever, Elton Carlos Costa
L658c Casos especiais ótimos de algoritmos aproximativos para problemas de escalonamento com restrições de precedência em processadores paralelos idênticos / Elton Carlos Costa Lever. 2017
61 f.: 31 cm.

Orientadora: Rosiane de Freitas Rodrigues
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. algoritmos aproximativos. 2. árvores de precedência. 3. processadores paralelos. 4. tempo unitário de processamento. 5. teoria de escalonamento. I. Rodrigues, Rosiane de Freitas II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Casos Especiais Ótimos de Algoritmos Aproximativos Para o Problema Clássico de Escalonamento de Tarefas com Restrições de Precedência e Tempo Unitário em Máquinas Paralelas Idênticas"

ELTON CARLOS COSTA LEVER

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Profa. Rosiane de Freitas Rodrigues - PRESIDENTE

Prof. Raimundo da Silva Barreto - MEMBRO INTERNO

Prof. José Luiz de Souza Pio - MEMBRO EXTERNO

Manaus, 22 de Junho de 2017

A Deus,
aos professores,
aos amigos de curso,
as minhas três meninas e
minha querida esposa.

Agradecimentos

Agradeço primeiramente a Deus, hoje e sempre, pois tenho certeza que ele me permitiu passar por esse desafio e sempre caminhou comigo.

A minha orientadora, Rosiane de Freitas Rodrigues, pela oportunidade de enriquecimento profissional e pessoal, dedicação, paciência e pela competência que sempre encaminhou essa pesquisa.

Aos meus amigos do grupo de otimização (Algox) Bruno Dias, Rainer Amorim, Sidney Lins, Simone Gama, Omar Latorre e Danielle Gordiano, que me deram suporte diretamente e indiretamente nessa trajetória.

A minha esposa Joseane Lever, as minhas filhas Andressa Lever, Adriele Lever e Vitória Lever pela paciência de ausência em alguns momentos. A minha mãe Graça Lever e meus irmãos Ede Lever, Noeme Lever, Leandro Potiguara, Eloise Lever e o caçula Sócrates Lever, o qual começou comigo nesse desafio.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo suporte financeiro por 2 anos.

Resumo da Dissertação apresentada ao PPGI/IComp/UFAM como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

CASOS ESPECIAIS ÓTIMOS DE ALGORITMOS APROXIMATIVOS PARA PROBLEMAS DE ESCALONAMENTO COM RESTRIÇÕES DE PRECEDÊNCIA EM PROCESSADORES PARALELOS IDÊNTICOS

ELTON CARLOS COSTA LEVER

Março/2017

Orientadora: Profa. Rosiane de Freitas , D.Sc.

Esta dissertação aborda a classe de problemas de escalonamento de tarefas com restrições de precedências e tempos unitários em processadores paralelos idênticos. Tal classe de problemas tem uma grande importância em teoria da complexidade computacional, uma vez que pequenas variações nas condições envolvidas no escalonamento, fazem com que um problema fácil se torne muito difícil. Dois grandes problemas envolvem a condição do número de processadores, onde, se o número de processadores for variável, dado como entrada, tal problema é provado ser NP-completo, mas, se o número de processadores for fixo, o problema ainda está em aberto. Neste contexto, o foco da pesquisa envolve o problema já provado ser NP-completo, onde para qual se investigou os principais algoritmos aproximativos existentes na literatura e suas provas de razão de aproximação do ótimo, tais como o algoritmo 2-aproximativo de Garey & Jonhson e as melhorias de Hu, Coffman & Graham e de Gangal & Ranade (GR) com $2 - \frac{7}{3P+1}$, o de melhor razão de aproximação da literatura. As provas de razão de aproximação de tais algoritmos foram detalhadas. Como principal contribuição da pesquisa, foram determinados casos especiais ótimos, para classes específicas de grafos direcionados acíclicos que envolvem arborescências (árvores de precedência, como *in-tree* e *out-tree*) para o melhor algoritmos aproximativo da literatura.

Palavras-chave: algoritmos aproximativos, árvores de precedência, processadores paralelos, tempo unitário de processamento, teoria de escalonamento, teoria dos grafos.

Abstract of Dissertation presented to PPGI/IComp/UFAM as a partial fulfillment of the requirements for the degree of Master in Informatics.

ELTON CARLOS COSTA LEVER

March/2017

Advisor: Prof. Rosiane de Freitas , D.Sc.

This dissertation addresses the class of job scheduling problems with precedence constraints and unit execution times, in identical parallel processors. Such a class of problems is of great importance in computational complexity theory, since small variations in the conditions involved in scheduling make an easy problem very difficult. Two major problems involve the condition of the number of processors, where, if the number of processors is variable, given as input, such problem is proved to be NP-complete, but if the number of processors is fixed, the problem is still open. In this context, the focus of the research involves the problem already proven to be NP-complete, where for which we investigated the main approximation algorithms in the literature and their proofs of approximation ratio of the optimal, such as of the Garey & Jonhson's 2-approximation algorithm, of the Hu, of the Coffman & Graham, and of the Gangal & Ranade with $2-3P+1$, the best approximation ratio in the literature. The approximation ratio proofs of such algorithms were detailed. As the main contribution of this research, were proved the optimality for specific classes of acyclic directed graphs involving trees (precedence trees, such as in-tree and out-tree) for the best approximation algorithms literature.

Keywords: approximative algorithms, graphs theory, parallel processors, precedence tree, processing unitary time, scheduling theory, .

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Problemas de escalonamento de tarefas constituem uma grande classe de problemas de muita importância teórica e que representam importantes aplicações práticas em diversos seguimentos de mercado que envolvem a alocação de recursos na realização de tarefas e sob algum critério de otimização em função do tempo. Os estudos nessa área tiveram início em meados dos anos cinquenta, logo após o surgimento dos primeiros computadores, e continuam sendo feitos cada vez mais intensamente nos dias atuais, constituindo a área da teoria de escalonamento.

Nesse contexto, problemas de escalonamento de tarefas que possuem restrições de precedência para suas execuções, é uma das mais importantes e desafiadoras classes de problemas de escalonamento, em que não é difícil encontrar cenários práticos de aplicação, como em diversos processos industriais, como em uma fábrica de refrigerantes, onde para um refrigerante ser produzido, algumas tarefas precedem outras, tal como a necessidade de encher o liquido da garrafa, antes de tampá-la. Tais problemas compartilham características entre si, sendo possível criar modelos teóricos genéricos capazes de abranger inúmeros casos de aplicação. Porém, quanto mais complexas forem essas modelagens, mais difícil se torna encontrar métodos capazes de escalonar tarefas otimamente. Muitos problemas em suas formulações genéricas foram provados pertencer à classe de problemas NP-completo, o que torna improvável a existência de soluções eficientes (em tempo polinomial). Por esta razão, muitas soluções envolvendo problemas de escalonamento são propostas para casos mais restritos, enquanto outras soluções são dadas de forma aproximada. Dois grandes problemas envolvem a condição do número de processadores, onde, se o número de processadores for variável, dado como entrada,

tal problema é provado ser NP-completo, mas, se o número de processadores for fixo, o problema ainda está em aberto (pertence a classe NP, mas não é provado se pertence a classe NP-difícil), que no caso de ser três processadores corresponde ao "*open 8*", oitavo problema da famosa lista de doze problemas em aberto do livro clássico sobre teoria da NP-completude [Garey and Johnson(1979)], e que é um dos dois únicos problemas de tal lista que permanecem em aberto. No outro caso, se o número de processadores for variável, dado como entrada na resolução do problema. Já é provado ser NP-difícil, pela redução do problema 3-SAT [Ullman(1975)] e pela redução do problema da clique máxima [J. K. Lenstra(1978)]. Ambos serão apresentados neste texto.

Nesta dissertação, o foco da pesquisa envolveu a classe de problemas de escalonamento de tarefas com restrições de precedências e tempos unitários de execução, em processadores ou máquinas paralelas idênticas, para minimizar o tempo de execução da última tarefa a ser escalonada (makespan). O problema geral desta classe, envolvendo o número de processadores, é NP-completo como afirmado acima. Neste caso, há diversos algoritmos heurísticos propostos, com analise de quantidade da solução gerada, indicando qual a distância do ótimo no pior caso, ou seja, com uma análise da razão de aproximação da solução ótima do problema. Para dois processadores apenas, o problema é polinomial e os algoritmos fornecem a solução ótima, mas, no caso de três ou mais processadores, apresentam soluções próximas do ótimo, como o algoritmo 2-aproximativo de Garey & Jonhson, o de Hu ($2-\frac{1}{|P|-1}$), o de Coffman & Graham ($2-\frac{2}{|P|}$), e finalmente Gangal & Ranade ($2-\frac{7}{3|P|+1}$), com melhor razão de aproximação. Existem resultados para casos especiais de restrição de precedência modeladas como um grafo direcionado acíclico (do inglês, DAG - *Directed Acyclic Graph*), como algoritmos ótimos para o problema com restrições de precedência em arborescências (árvores de precedência), mas, para o algoritmos de melhor razão de aproximação este estudo de casos especiais ótimos não existia, e se realizou nesta presente pesquisa.

Como contribuição desta pesquisa, será apresentada uma abordagem analítica sobre algoritmos aproximativos que se aplicam ao problema $P|prec, p_j = 1|C_{max}$, onde $|P|$ é a quantidade de processadores variáveis, p_j é o tempo de processamento de cada tarefa, *prec* são as restrições de precedências gerais e C_{max} consiste na minimização do makespan. Por consequência é explicado também a dedução da razão de aproximação de tais algoritmos e, principalmente, é provada a otimalidade do principal algoritmo aproxima-

tivo da literatura para restrições de precedência que representam arborescências (árvores de precedência).

1.1 Motivação e justificativa

Desde a revolução industrial até os dias atuais o número de pesquisadores que buscam melhorar e diminuir o tempo de escalonamento total das tarefas em processadores paralelos tem aumentado de maneira significativa, tornando necessária a expansão do conhecimento dentro das universidades em parceria com empresas, utilizando-se de soluções computacionais com algoritmos aproximativos dentre os quais são estudados nesta pesquisa. Além dos problemas que modelam situações práticas, muitos problemas teóricos sobre escalonamento também são alvo de pesquisa por meio de ferramentas existentes na literatura.

1.2 Objetivos

Os objetivos desta pesquisa estão proposto da seguinte forma:

1.2.1 Objetivo geral

O principal objetivo desta pesquisa consiste em determinar casos especiais ótimos do algoritmo de melhor razão de aproximação da literatura para o problema de escalonamento com tarefas unitárias e precedências, em processadores paralelos idênticos.

1.2.2 Objetivos específicos

Os objetivos específicos estão propostos como segue:

- identificar, analisar e reescrever os principais algoritmos aproximativos da literatura para o problema em estudo;
- detalhar a prova da razão de aproximação do ótimo, para estes principais algoritmos aproximativos;
- demonstrar casos especiais ótimos, para subclasses de grafos direcionados acíclicos, em especial arborescências, destes principais algoritmos aproximativos.

1.3 Estrutura da dissertação

O restante deste documento é organizado da seguinte forma: No Capítulo 2, são introduzidos alguns conceitos teóricos utilizados na literatura em otimização combinatória e também é apresentado o problema de escalonamento com restrição de precedência e tempo unitário das tarefas atribuídas em processadores paralelos idênticos fixos e variáveis. No Capítulo 3, é mostrado os algoritmos aproximativos e suas deduções para a razões de aproximação do ótimo de cada um deles. No Capítulo 4, são mostrados os resultados sobre análises e as comparações das razões de aproximação dos algoritmos apresentados, assim como uma prova de otimalidade do algoritmo de Gangal & Ranade para árvores com precedência. Por fim, no Capítulo 5, são feitas as conclusões.

Capítulo 2

Fundamentos teóricos

Os algoritmos de otimização combinatória se mostram de grande interesse para a computação. Eles permitem a análise e otimização de situações em diversas áreas de aplicação. Alguns poucos exemplos são: redes sociais, mapeamento genético, rotas de aviação, sistemas de comunicação, escalonamento de tarefas, gerenciamento de estoque, bancos de dados, entre muitos outros. Nessa pesquisa científica, propôs-se a abordagem de algoritmos que abrangem teoria dos grafos, teoria de escalonamento e algoritmos de aproximação. Além disso, foi necessária uma familiarização com linguagens de programação e com estrutura de dados. É de extrema importância o conhecimento dos algoritmos de otimização e suas heurísticas quando se quer gerar soluções aproximadas. No entanto, em diversas aplicações apenas o uso destes algoritmos não é suficiente. É necessário antes modelar a situação e adaptá-la para o uso de algoritmos conhecidos. Essa tarefa, na maioria das vezes, é bastante difícil, por isso torna-se mais interessante para pesquisadores na ciência da computação na área de otimização.

2.1 Teoria dos grafos

Um grafo é um par ordenado $G = (V, E)$, onde V é um conjunto de vértices e E é um conjunto de arestas, onde $n = |V|$ e $m = |E|$. Cada elemento e no conjunto E é um par (i, j) que indica que o vértice i é ligado ao vértice j (ou seja, são adjacentes, e a aresta e incide em i e j). O grafo é dito direcionado quando os pares que representam as arestas são ordenados, isto é, $(i, j) = (i \prec j)$ isso quer dizer que i deve ser executado e finalizado antes que j inicie seu processo de execução. A representação gráfica de um grafo consiste

em pontos distintos do plano associados a cada vértice e para cada aresta (i, j), uma seta conectando os pontos correspondentes aos vértices i para j.

O grau de um vértice v do grafo equivale à quantidade de arestas que incidem em v. O grau máximo de um grafo, denotado por $\Delta(G)$, é o valor do maior grau dentre todos os vértices de G. De maneira similar, o grau mínimo, denotado por $\delta(G)$, é definido como o valor do menor grau de G. Um grafo completo é aquele no qual cada vértice é vizinho de todos os demais, ou seja, o grau de todos os vértices é exatamente igual a $n-1$, portanto a quantidade de arestas de G é dada então por $m = \frac{n(n-1)}{2}$. Um exemplo de grafo completo é dado na Figura 2.1.

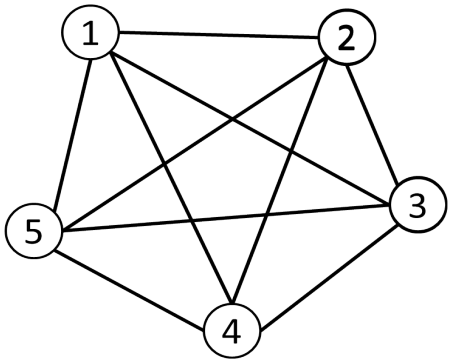

Figura 2.1: Exemplo de grafo completo com 5 vértices, conhecido como K_5, onde cada vértice tem grau 4.

Um problema clássico cujas instâncias podem ser modeladas em grafos é o problema de escalonamento com restrição de precedência e tempo unitário das tarefas, onde: Seja n o número de tarefas (job) e $|P|$ o número de processadores, assim, dado um conjunto S de n tarefas, uma relação de precedência em S, e cada tarefa tendo tempo unitário de processamento $(p_j = 1)$ em uma série de processadores paralelos $|P|$. As tarefas devem obedecer uma relação de precedência $\prec$, onde $j \prec j'$, o que significa que a tarefa j precede a tarefa j' ou seja, que a tarefa j' deve ser executada somente depois que a tarefa j tiver sido finalizada em algum processador. Esse tipo especial de conjunto parcialmente ordenado (POSet) podem ser divididos em dois casos.

No primeiro caso o grafo pode ter ciclos que se caracteriza ter circuito, contudo não será foco de estudo nesse trabalho. A Figura 2.2 mostra um exemplo de de um grafo direcionado que contem ciclos.

No segundo caso o conjunto parcialmente ordenado (POSet) é um grafo acíclico direcionado ***Directed Acyclic Graph*** (DAG), que é um tipo peculiar de árvore que pode ter, entre outras, as seguintes características:

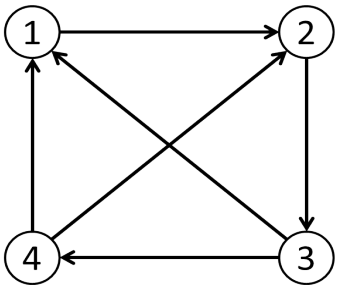

Figura 2.2: Exemplo de grafo direcionado cíclico com 4 vértices.

- ***In-tree:*** Todos os vértices tem apenas 1 sucessor com exceção da raiz que não tem nenhum. (Consiste em uma árvore que a direção dos arcos vai a partir dos nós em direção à raiz e cada nó, com exceção da raiz, tem um único sucessor).

- ***Out-tree:*** Todos os vértices tem apenas 1 antecessor com exceção da raiz que não tem nenhum. (Consiste em uma árvore que a direção dos arcos vai a partir da raiz em direção aos demais nós, e que cada nó, com exceção da raiz, tem apenas um antecessor).

- ***In-forest:*** é um conjunto de duas ou mais *in-tree*.

- ***Out-forest:*** é um conjunto de duas ou mais *out-tree*.

- ***Opposing forest (OF):*** é quando existe pelo menos uma *in-tree* e uma *out-tree* juntas desconexas.

- ***Chain:*** é quando um vértice liga-se a no máximo um único vértice adjacente, ou seja, tendo no máximo um predecessor e um sucessor, sendo que o primeiro não tem predecessor e o último não tem sucessor. Para este caso especifico as cadeias (*chains*) é um caminho.

A Figura 2.3 mostra alguns exemplos de DAGs que serão usados como exemplo no decorrer deste trabalho, entre outros. Para reforçar ainda mais sobre esse tipo especial de grafo mais algumas definições será útil.

Outros exemplos de DAGs, mas que servirão somente para efeito de informação neste trabalho, são os que estão mostrado a seguir e exemplificados na Figura 2.5.

- ***Interval orders:*** não tem uma subestrutura isomorfa ao tipo I da Figura 2.5.

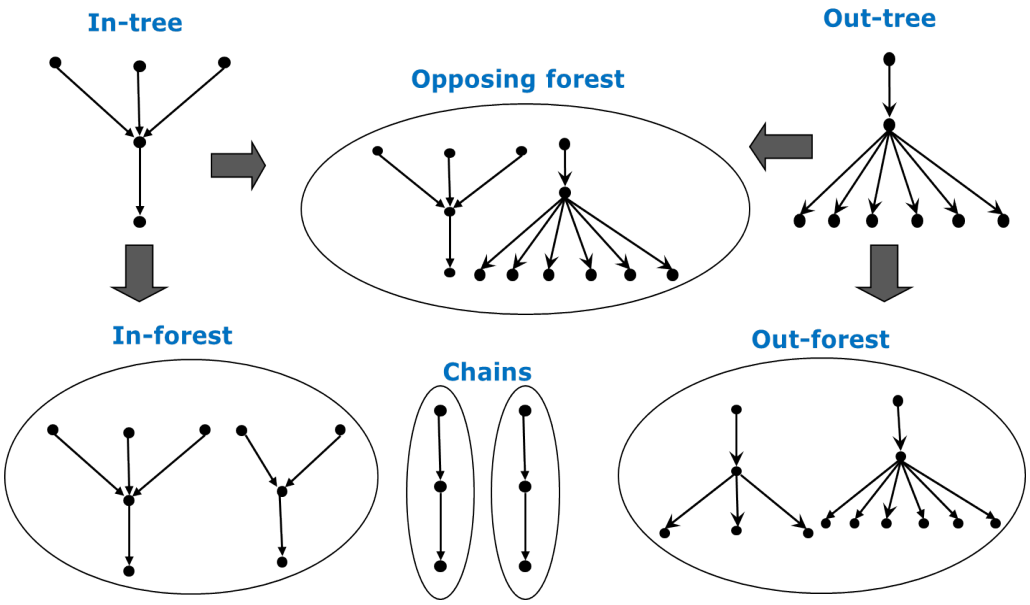

Figura 2.3: Exemplo de DAGs (mostra as arvores enraizadas in-tree, out-tree, in-forest, out-forest, opposite forest e chains).

- ***Quasi-interval orders:*** não tem uma subestrutura isomorfa aos tipos II, III ou IV da Figura 2.5.

- ***Over-interval orders:*** não tem uma subestrutura isomorfa aos tipos II e III da Figura 2.5.

- ***Series-parallel orders (SP):*** Uma ordem de precedência é (SP) se seu DAG G correspondente de somente uma tarefa ou puder ser dividido em 2 grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ tal que todas as tarefas ou vértices de G_1 devam ser concluídas antes das tarefas de G_2 iniciarem a execução ou não existam restrições de precedência entre quaisquer duas tarefas de grafos distintos. A Figura 2.4 letra "a" mostra um exemplo de grafo que não é (SP). No entanto o tipo I da Figura 2.5 poderá ser (SP) se e somente se (a,c) forem executados primeiro que (b,d).

- ***Level orders:*** Quando cada componente conexo de seu DAG correspondente puder ser particionado em l níveis (levels), tal que para cada escolha de tarefas, $j_i \in l_l \;\; e \;\; j'_i \in l_{l-1}$, $j_i \prec j'_i$. Um exemplo é mostrado na letra "b" da figura 2.4.

Em uma DAG qualquer existem algumas propriedades que serão mostradas a seguir: Seja $j_i \prec j_{i+1} \prec j_{i+2}$ então:

- j_i é predecessor imediato de j_{i+1}, por consequência j_{i+1} é sucessor imediato de j_i;

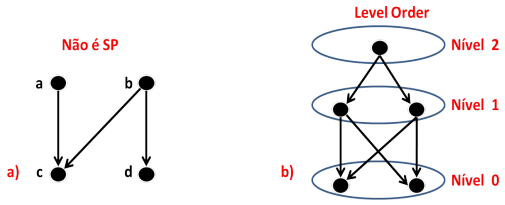

Figura 2.4: Exemplo de DAGs (mostra um grafo que não é Series-parallel orders (SP)).

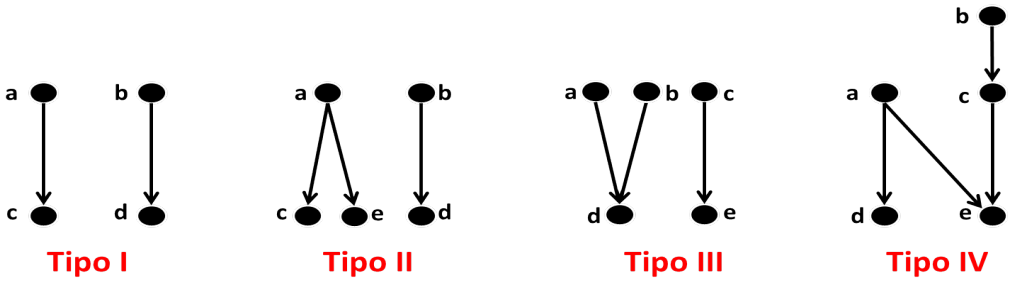

Figura 2.5: Exemplo de DAGs (mostra os grafos do tipo I, II, III e IV).

- j_i é predecessor de j_{i+2} e j_{i+2} é sucessor de j_i por transitividade.
- Se houver um arco que liga de j_i diretamente a j_{i+2}, então esse arco pode ser descartado por redução do fecho transitivo sem prejudicar a precedência essencial do grafo. Um exemplo é mostrado na Figura 2.6. Em "a" o grafo inicial e em "b" o grafo com a redução de transitividade.

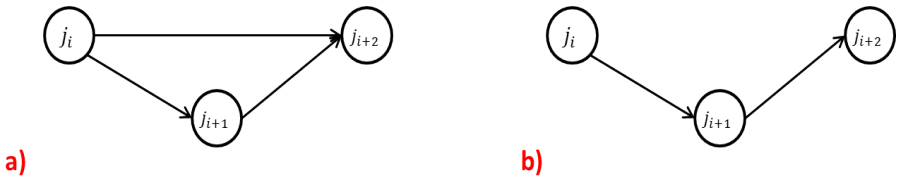

Figura 2.6: Exemplo um grafo com 3 vértices sendo executada a redução do fecho transitivo.

2.2 Teoria de escalonamento

Um problema de escalonamento envolve, em sua forma mais comum, um conjunto de n tarefas, chamado J que devem ser executadas em um conjunto de $|P|$ processadores. Tanto as tarefas quanto os processadores podem ter variações que caracterizam os inúmeros problemas de escalonamento conhecidos. As tarefas podem ter, entre outras, as seguintes características:

2.2.1 Características das tarefas

Cada **tarefa** (em inglês, ***job*** ou ***task***) pode ter algumas restrições associadas, como segue:

- **Tempo de processamento** (p_j): Indica o tempo que a tarefa deve permanecer sendo executada nos processadores (geralmente, um) para que a mesma seja cumprida;
- **Data de Disponibilidade (*release date*)** (r_j): É o menor instante no qual a tarefa pode começar a ser executada;
- **Prazo Recomendado de Término (*due date*)** (d_j): Tempo recomendado de completude da tarefa. Se a mesma terminar depois deste prazo, haverá uma penalidade;
- **Prazo de Término (*deadline*)** ($\overline{d_j}$): Prazo para a completude da tarefa (não pode ser ultrapassado);
- **Peso** (w_j): Um valor que pode ser aplicado como uma penalização da tarefa em cima de algum critério (como violação de *due date*, por exemplo).

2.2.2 Representação e notação

Com o intuito de facilitar a referência a um tipo específico de problema, foi introduzida a **notação de três campos** por [Graham et al.(1979)Graham, Lawler, Lenstra, and Kan], cujo formato é $\alpha \mid \beta \mid \gamma$, que a seguir será mostrada as características mais importantes para este trabalho:

- α é o ambiente de processamento, sendo alguns valores possíveis para este campo:
  - **1**: Há apenas um processador ou máquina disponível para todas as tarefas;
  - **P**: Existem $|P|$ processadores paralelos idênticos disponíveis, onde $|P|$ é um parâmetro de entrada para o problema;
  - **Q**: Há $|P|$ processadores paralelos uniformes de velocidades diferentes, sendo q_i a velocidade do i-ésimo processador. Devido às velocidades diferentes, tem-se que o tempo que a j-ésima tarefa passa no i-ésimo processador (considerando que j será executada por completo em i) é $\frac{p_j}{q_i}$.
- β é um conjunto de características das tarefas e suas restrições (se houver), sendo algumas mais importantes para esse trabalho as listadas a seguir:

- **prec**: Indica restrição de precedência entre tarefas, que são conjuntos parcialmente ordenados (POSet) especialmente DAGs. Ver Figura 2.3;
- r_j: Se este valor estiver presente, as tarefas possuem *release date*;
- p_j: Indica o tempo de processamento que cada tarefa tem, sendo que, por exemplo, $p_j = 1$ indica que cada uma das tarefas tem tempo de processamento igual a 1;
- d_j: As tarefas possuem *due dates*;
- $\overline{d_j}$: As tarefas possuem prazos de términos obrigatórios *(deadlines)*.

- γ é a função objetivo a ser otimizada. Alguns valores são usados por tais funções nessa classe de problemas, tais como:
  - **Tempo Total de Completude (*total completion time* - C_j)**: O instante no qual a tarefa terminou sua execução, sendo dado por $C_j = s_j + p_j$, onde s_j é o instante de início da tarefa;
  - **Latência (*lateness* - L_j)**: A distância temporal entre o instante em que a tarefa foi completada e o seu *due date*. Assim, $L_j = C_j - d_j$;
  - **Atraso (*tardiness* - T_j)**: Quanto tempo a tarefa foi executada após o seu *due date*. Temos então $T_j = max\{0, L_j\}$.

  Tem-se então algumas das principais funções a serem otimizadas:

  - ***Makespan* - C_{max}**: Minimização do maior tempo de completude dentre todas as tarefas executadas. C_{max}, portanto, é a extensão do tempo usado para executar todas as tarefas;
  - **Latência máxima - L_{max}**: Minimização da maior latência dentre todas as tarefas. Deseja-se, então, terminar as tarefas o mais cedo possível;
  - **Tempo de Completude Ponderado - $\sum w_j C_j$**: Minimização da soma dos tempos de completude das tarefas, sendo cada tempo multiplicado pelo peso w_j da tarefa (se $w_j = 1$, temos a versão não ponderada);
  - **Atraso Ponderado - $\sum w_j T_j$**: Minimização da soma dos atrasos das tarefas com os pesos w_j das mesmas aplicados (novamente, (se $w_j = 1$, temos a versão não ponderada).

Para a abordagem desta pesquisa terá foco somente o problema de escalonamento com restrição de precedência em processadores paralelos idênticos com tarefas de tempo de processamento unitário que, dentre muitas, terá o foco maior em DAGs ou POSet com ou sem redução de transitividade *int-tree, out-tree, in-forest, out-forest, chains* e *opposite forest* ou até mesmo as arbitrárias que serão escalonadas de acordo com cada algoritmo usado sem a utilização de preempção.

O problema de escalonamento com restrição de precedência e tempo unitário das tarefas em processadores paralelos fixos consistem em: Seja n o número de tarefas (*jobs*) a serem escalonadas em $|P|$ processadores paralelos e idênticos, então dado um conjunto S de n tarefas, uma relação de precedência em S, e cada tarefa tendo tempo unitário de processamento $(p_j = 1)$ para serem executadas em uma série de processadores $|P|$, tendo como objetivo minimizar o tempo total de completude.

2.3 Noções de otimização combinatória

Um problema de otimização consiste em encontrar a melhor solução dentre um conjunto de candidatos. A qualidade da resolução é medida por meio de uma **função objetivo**, sendo que em um problema de minimização, a melhor solução será aquela de menor valor, e em um problema de maximização, a solução será a de maior valor [Papadimitriou and Steiglitz(1998)]. O formato básico de um problema de otimização é:

$$\textbf{Minimizar / maximizar } z = cx$$
$$\textbf{Sujeito a } x \in P$$
$$x \in \mathbb{R}^+$$

Onde x é uma solução, $f(x)$ é a função objetivo e P é o conjunto de soluções. Este conjunto é definido pelas características e restrições de cada problema. O tamanho deste conjunto pode ser contável ou não, sendo que no primeiro caso, o problema é dito ser de otimização combinatória.

Na área de Pesquisa Operacional os problemas são representados por modelos matemáticos que é representação simplificada de uma situação da vida real, formalizado com símbolos e expressões matemáticas. Os tipos de modelos matemáticos em otimização são:

- Programação Linear Contínua e Inteira;
- Programação Não-linear.

Na programação linear continua, um caso particular dos modelos em que as variáveis são contínuas e apresentam comportamento linear, tanto em relação às restrições como à função objetivo. Uma ferramenta bastante utilizada em problemas de otimização é a programação linear. Nela, a função objetivo é linear (ou seja, um polinômio de grau 1), e o conjunto de soluções é definido por restrições também lineares.

Um problema de programação não-linear é parecido com um de programação linear, porém a diferença é que alguma das funções descritas (função objetivo ou de restrições) pode ser não-linear podendo estar incluso na forma quadrática, cúbica ou de maior grau.

2.4 Métodos aproximados: heurísticas e meta-heurísticas

Em função da elevada complexidade computacional necessária na determinação de uma solução exata para uma grande quantidade de problemas combinatórios (denominados problemas NP-difíceis), muitos algoritmos heurísticos propostos podem não achar diretamente uma solução ótima global. Nesses casos, adota-se normalmente uma estratégia que vise equilibrar a qualidade da solução obtida tal como tempo total de processamento. Isso pode ser viabilizado, por exemplo, através de métodos aproximativos que mostram por meio de uma razão de aproximação o quão distante uma solução está do ótimo.

Uma das limitações dos métodos que visam a determinação de soluções ótimas para problemas combinatórios é que, raramente, eles resolvem grandes instâncias em um tempo computacional aceitável. Em função disso, muitos pesquisadores se concentraram no desenvolvimento de heurísticas ou meta-heurísticas inteligentes. Estas técnicas se enquadram dentro de uma categoria especial mais conhecida na literatura como heurísticas ou métodos inteligentemente flexíveis (Redes Neurais, Algoritmos Genéticos, etc). Trata-se de estratégias que buscam, da melhor maneira possível, fazer uso de técnicas já existentes como heurísticas especializadas juntamente com um conhecimento prévio do problema no intuito de chegar a um desempenho algorítmico mais eficiente. Na verdade, pode-se afirmar que grande parte dos problemas combinatórios de interesse prático são

NP-difíceis, Isto significa que a possibilidade de resolvê-los em tempo polinomial não está totalmente descartada.

2.4.1 Algoritmos aproximativos

Na ciência da computação existe algumas classes de problemas onde é difícil provar se existe algoritmo que resolva de forma ótima um determinado problema e muitos na literatura já foram provados ser NP-completo, contudo existem heurísticas e ou meta-heurísticas que apresentam técnicas que dizem o quão próximo do ótimo está a solução de um problema e isso se mostra com algoritmos aproximativos apresentando a razão de aproximação deles em relação ao ótimo.

Quatro algoritmos serão mostrados nessa pesquisa com enfoque para escalonamento de tarefas com restrição de precedência e tempo unitário de processamento em processadores paralelos idênticos, sendo três deles aproximativos e um eficiente.

Nesta pesquisa adota-se técnicas de construção de algoritmos aproximativos existentes na literatura, os quais apresentam um limite de distância do ótimo, mas que dependendo da classe específica de instância pode dar solução ótima, onde no Capítulo [3] será abordado e apresentado algoritmos com mais detalhes.

Segundo [Cormen et al.(2009)Cormen, Leiserson, Rivest, and Stein] algoritmo aproximativos são usados para encontrar soluções aproximadas em problemas de otimização, que adota como entrada não somente uma instância do problema, mas também um valor $\varepsilon > 0$, tal que para ε fixo, o esquema é um algoritmo $(1+\varepsilon)$ - aproximativo.

Um algoritmo aproximativo para um determinado problema é um algoritmo que gera soluções aproximadas para tal problema. Ele obtêm um limite razão entre a solução ótima e a produzida pelo algoritmo aproximado.

2.5 Problema de escalonamento de tarefas com restrição de precedência e tempo unitário envolvendo processadores paralelos

O problema de escalonamento de tarefas com restrição de precedência, tendo tempo de execução unitário com $|P|$ processadores para executar todas as tarefas tem sido extensi-

vamente estudado. O problema foi demonstrado ser NP-completo quando o número de processadores é parte da entrada, ou seja, quando o número de processadores são variáveis $|P|$ sendo $P|prec, p_j = 1|C_{max}$ [Ullman(1975)] e [J. K. Lenstra(1978)]. Du et al. provou que $P_2|chains|C_{max}$ é fortemente NP-difícil [J. Du(1991)]. Garey e Jonhson apresentaram um algoritmo eficiente que dá o ótimo quando $|P| = 2$ [Garey and Johnson(1976)]. Foi demonstrado também por [Coffman Jr. and Garey(1993)] que, para o caso de dois processadores pode ser resolvido em tempo polinomial tendo preempção ou não. A complexidade do correspondente problema em k processadores fixos ainda continua em aberto na teoria de NP-completude.

O problema $P|prec, p_j = 1|C_{max}$, onde $|P|$ representa a quantidade de processadores ou máquinas variáveis não pode ser confundido com $P_k|prec, p_j = 1|C_{max}$, onde k é fixo e a diferença está exatamente que um é variável (parâmetro) e outro é fixo. [Ullman(1975)] fez a redução ao problema de 3 Satisfabilidade, no entanto para processadores fixos seria necessário elaborar prova para cada quantidade de processadores, pois nesse caso, a quantidade não seria parâmetro de entrada e sim estaria definida dentro do algoritmo junto a prova.

2.5.1 Caracterização de cenário com número fixo de processadores

O problema $P|prec, p_j = 1|C_{max}$ em que os processadores são fixos continua em aberto. Se a quantidade de processadores for igual a 3, então o problema $P_3|prec, p_j = 1|f(x)$ é conhecido como "Open 8" por ser o 8^o problema em aberto da famosa lista escrita no livro dos autores [Garey and Johnson(1979)].

Considera-se o escalonamento como um conjunto de tarefas com ordenação parcial entre elas em um conjunto de processadores paralelos para executar essas tarefas e que cada tarefa está associada a um processador disponível em algum momento. Em geral, nem todos os processadores estarão disponíveis para cada tarefa, dependendo da restrição de precedência entre elas. Portanto, uma tarefa só pode ser processada uma única vez em algum processador do conjunto de todos processadores. Finalmente tem-se a restrição de precedência, onde, existindo duas tarefas quaisquer j e j', a relação de precedência é dada por $j \prec j'$, significa dizer que j' só poderá começar a ser executada após o término da tarefa j, portanto o tempo de escalonamento da tarefa j deve ser menor que o tempo da tarefa j' nesse mesmo escalonamento.

Características do problema $P_k|prec, p_j = 1|C_{max}$:

- O primeiro campo (P_k) significa que o ambiente é de máquinas ou processadores paralelos e idênticos com um número k fixo.
- O segundo campo ($prec, p_j = 1$) consta a palavra *prec* o qual significa que as tarefas têm relação de precedência entre elas, ou seja, que uma depende da outra para ser escalonada. Nesse mesmo campo tem $p_j = 1$, que significa o tempo de processamento de cada tarefa em qualquer processador disponível, que nesse caso é exatamente igual a um.
- O terceiro campo (C_{max}) há a função objetivo C_{max}, que significa diminuir o *makespan*, ou seja, ter o máximo na diminuição do tempo de completude de todas as tarefas escalonadas. Quando a última tarefa terminar de ser executada, será definido o tempo de completude total do escalonamento das tarefas que se deseja minimizar.

A representação das instâncias para esse problema é dado por meio de um Grafo Direcionado e Acíclico (DAG), onde cada vértice representa uma tarefa e cada aresta direcionada (arco) é a representação da relação de precedência entre as tarefas mencionado na Seção 2.1.

Para um número fixo de processadores a instância de entrada é somente o grafo, pois a quantidade de processadores já estará disponível no algoritmo.

2.5.2 Caracterização de cenário com número variável de processadores

Em um cenário com número variável de processadores o problema consiste em dizer que, assim como as instâncias, os processadores ou máquinas também fazem parte das entradas como parâmetro, ou seja, o algoritmo para esse problema não precisa ser configurado com o total de processadores para executar as tarefas, pois essa quantidade será passada como parâmetro ao escalonamento.

Na notação de três campos o problema é caracterizado como: $P|prec, p_j = 1|C_{max}$, o objetivo minimizar o tempo total de processamento das tarefas e sua complexidade é NP-completo [Ullman(1975)], contudo alguns pesquisadores propuseram algoritmos aproximativos que garante uma razão de aproximação.

As Figuras 2.7 e 2.8 ilustram o que foi dito sobre processadores fixos e varáveis. Nelas há uma representação por matriz de adjacência no cenário de escalonamento com a utilização de processadores que irão executar as tarefas.

Seja um grafo arbitrário proposto pela matriz da Figura 2.7 representado pela Figura 2.8.

0	1	1	0	0	0	1	0
0	0	0	1	1	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	1	1
0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figura 2.7: Matriz de adjacência para o exemplo de grafo arbitrário utilizado para o escalonamento com restrições de precedência em $|P| = 3$.

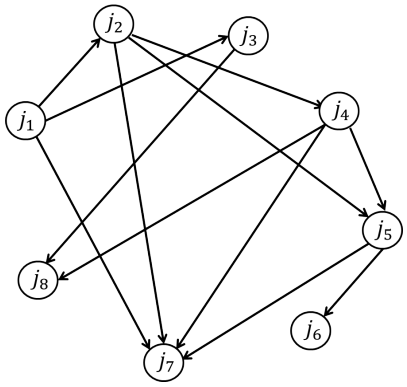

Figura 2.8: Grafo gerado da matriz de adjacência da Figura 2.7.

No grafo da Figura 2.8 a tarefa j_1 é pai de todas as outras e j_2, j_3 e j_7 são suas sucessoras imediatas, contudo j_7 não pode ser escalonada no mesmo instante de j_2 e j_3 por conta de ser sucessora de j_2, assim se aplica em todo grafo até as folhas, pois não têm sucessores.

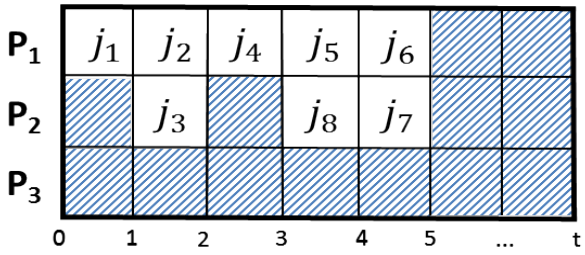

Figura 2.9: Escalonamento genérico do grafo da Figura 2.8.

Executando o escalonamento em três processadores, para demonstrar como representa tal escalonamento dessa instância, levando em consideração que nesse momento não será

usado um algoritmo especifico, somente será escalonado por um genérico, contudo dentro das regras e restrições do problema. A Figura 2.9 é mostrada uma solução de escalonamento para a instância em $|P| = 3$, obedecendo as restrições de precedência, porém sem critério de otimização.

Pela linha do tempo pode-se verificar que as restrições de precedências estão obedecidas como também o tempo de processamento que cada tarefa tem. Assim, percebe-se que o processador 3 fica ocioso todo tempo e isso acontece porque, nesse grafo específico as restrições de precedência não liberaram a quantidade $|P|$ de tarefas em nenhuma coluna, ou seja, o escalonamento dessa instância não melhoraria em nada se $|P| > 2$, portanto, nesse caso para $|P| = 2$ apresenta o ótimo.

A Tabela 2.1 mostra resultados de complexidades com alguns problemas de escalonamento incluindo o desse estudo.

Tabela 2.1: Tabela de resultados de complexidade.

Problemas	**Complexidade**	**Autor/Ano**
$P\|tree, p_j = p\|C_{max}$	P	Hu [1961]
$P\|out-tree, p_j = p\|C_{max}$	P	Hu [1961]
$P\|\|C_{max}$	P	Graham [1966]
$P\|prec, p_j = p\|C_{max}$	P	Coffman e Graham [1972]
$P_k\|prec, p_j = p\|C_{max}$	Em aberto	————
$P_3\|prec, p_j = p\|f(x)$	Em aberto	————
$P\|prec, p_j = 1\|C_{max}$	NP-difícil	Ullman [1975]
$P_2\|\|C_{max}$	NP-difícil	Lenstra et al. [1977]
$P\|prec, p_j = p\|C_{max}$	NP-difícil	Lenstra et al. [1978]
$P\|chains\|\sum C_j$	NP-difícil	Du et al.[1991]

Onde:

$(|P|)$ é o número de processadores variáveis;

(k) são processadores fixos;

(p_j) é o tempo de processamento de cada tarefa em algum processador.

2.6 Resumo do capítulo

Neste capítulo foi apresentado uma revisão da literatura envolvendo os fundamentos teóricos necessários para a compreensão fundamental em teoria dos grafos, teoria de esca-

lonamento e otimização combinatória, incluindo noções de algoritmos aproximados. Em teoria dos grafos foi mostrado classes específicas de grafos com precedência em árvores. Em teoria de escalonamento foram resumidos os principais conceitos, complexidades, caracterização e notação. Foi explicado também sobre o cenário de processadores fixos e variáveis e suas diferenças. O estudo e familiarização desses conceitos teóricos foram de extrema importância para o desenvolvimento desta pesquisa.

Capítulo 3

Algoritmos para $P|prec, p_j = 1|C_{max}$ e suas razões de aproximação

Neste capítulo serão apresentados alguns algoritmos aproximativos relacionados ao problema em estudo, assim como a prova de dedução da razão de aproximação de três dos quatro algoritmos deste trabalho, mas antes será reforçado o que já foi explicado minuciosamente na Seção 2.5 que quanto ao número de processadores o problema é classificado em dois casos:

1. Se o número de processadores é uma variável $|P|$, o problema $P|prec, p_j = 1|C_{max}$ é NP-completo provado por [Ullman(1975)] que fez redução polinomial do 3-SAT, assim como Lenstra & Kan fizeram a redução da clique [J. K. Lenstra(1978)].

2. Se o número de processadores for um k fixo então $P_k|prec, p_j = 1|C_{max}$ está em aberto.

Não se sabe ainda qual a complexidade do problema $P_k|prec, p_j = 1|C_{max}$, pois para isso teria que ter um algoritmo para a prova que resolvesse a cada k fixo. Não é conhecido também se esse problema está na classe NP-difícil, pois para isso alguém terá que provar por meio de uma redução polinomial de um problema já conhecido ser NP-completo.

Primeiramente será citada algumas classes de instâncias para o problema proposto que já foram apresentadas em 2.1 em Grafos Direcionados Acíclicos (DAGs) (*in-tree, out-tree, In-forest, Out-forest, Opposite forest e Chain*), o qual é um conjunto parcialmente ordenado.

Será definido, para essa pesquisa, alguns conceitos, tais como:

- *Slot*: é um conjunto de vértices ou tarefas de um grafo que podem estar na mesma

coluna, nível ou ter o mesmo prazo a ser escalonado;

- Coluna: é uma unidade de tempo no escalonamento cuja a quantidade de vértices ou tarefas não ultrapasse o número de $|P|$ processadores disponíveis.

- Bloco: é um subgrafo 2-conexo maximal ou um subgrafo maximal que finaliza com um vértice de corte e tem um conjunto de *slots*;

- Segmento: é um bloco ou conjunto de blocos, também podendo ser um conjunto de $slots$ contíguos.

Os quatro algoritmos apresentados neste trabalho serão organizados por ordem cronológica, sendo três deles aproximativos e um eficiente, levando em conta a quantidade processadores $|P| \geq 3$, pois para $|P| = 2$ ambos algoritmos dão solução ótima.

Em 1961 Hu propôs um algoritmo de nivel por nível que utiliza a técnica Highest Level First (HLF)[Hu(1961)]. Em 1972, [Coffman Jr and Graham(1972)] refinaram o algoritmo de Hu dando uma abordagem alternativa que leva a um algoritmo $O(|V| + |E|)$. Trinta e cinco anos mais tarde, precisamente em 2008, [Gangal and Ranade.(2008)] deram um algoritmo que tem melhor aproximação do ótimo dentre os demais conhecidos acima e que aqui nesse trabalho terá atenção maior, portanto esses algoritmos serão destaque nesta pesquisa e a explicação de como procedem se encontram a seguir.

3.1 Algoritmo aproximativo de Hu (Highest Level First (HLF))

O algoritmo pioneiro foi proposto em 1961, por T.C. Hu e mostrou que resolve o problema $P|prec, p_j = 1|C_{max}$ em tempo polinomial quando as restrições de precedência das tarefas são árvores enraizadas (*in-tree, out-tree, in-forest e out-forest*) [Hu(1961)], [Papadimitriou and Yannakakis(1979)] consideraram a essas instâncias a complexidade na ordem de $O(|V|log|V|)$ segundo [Zang(2005)], onde V é o número de vértices, assim o algoritmo oferece um escalonamento ótimo quando $|P| = 2$, porém para $|P| \geq 3$ o algoritmo tem razão de $2 - \frac{1}{|P|-1}$, onde $|P|$ é o número de processadores.

Algoritmo 1: Algoritmo de Hu (HLF)

início

Atribuir um nível h a cada tarefa;

enquanto *houver tarefa sem nível* **faça**

se *a tarefa não tiver sucessores* **então**

$h(j) \leftarrow 1$

senão

$h(j) \leftarrow 1$ + *(nível máximo dos sucessores imediatos da tarefa atual)*;

fim se

fim enqto

Definir uma lista de prioridades pela ordem não crescente dos níveis das tarefas.

Executar uma estratégia de escalonamento baseando na lista de prioridades L.

fim

[Chen and Liu(1975)] mostraram que a razão de aproximação do algoritmo de Hu para o problema com restrições gerais de precedência é:

Para $|P| = 2$, o problema $P|prec, p_j = 1|C_{max}$ tem razão de aproximação 1 como mostrado na Tabela 4.1, portanto ótimo.

Para $|P| \geq 3$, sua razão de aproximação dista $2 - \frac{1}{|P|-1}$ do ótimo [Chen(1975)]. Esse algoritmo é conhecido por utilizar a estratégia de nível por nível e o caminho crítico usando uma lista para escalonar de maneira a pegar os de maior nível primeiro "*highest-level-first*"(HLF).

Basicamente o algoritmo organiza cada tarefa em nível utilizando uma escala onde a tarefa sem sucessor ficará no nível 1 e os seus predecessores imediatos ficam no nível 1 mais o nível maior de seus sucessores imediatos. Em seguida, organiza as tarefas em ordem não crescente dos níveis utilizando uma lista de prioridades em que as tarefas de maior nível serão escalonadas primeiro alocando nos primeiros processadores disponíveis até chegar na última tarefa que tem o menor nível, pois são justamente essas tarefas que não têm sucessores.

Teorema 1: o algoritmo de Hu apresenta uma razão $2 - \frac{1}{|P|-1}$ aproximação, onde $|P| \geq 3$ para o problema $P|prec, p_j = 1|C_{max}$.

Prova: Seja j e j' duas tarefas distintas e $t(j)$ é o tempo em que j é executada no escalo-

namento de Hu, então se $t(j) \leq t(j')$ implica que $\phi(j) > \phi(j')$. Onde ϕ é a prioridade de cada tarefa na lista L. A Figura 3.1 mostra um exemplo.

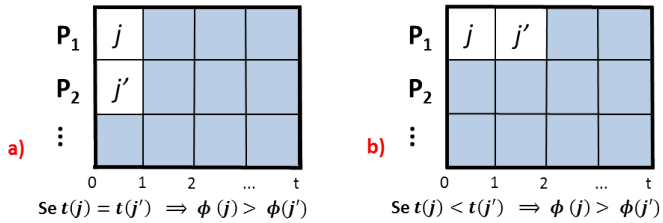

Figura 3.1: Exemplo da alocação de duas tarefas com relação de precedência no escalonamento usado pelo algoritmo de Hu.

Seja k uma tarefa paralela a j tal que $\phi(j) < \phi(k)$, contudo j está sendo escalonado em $t(j) < t(k)$, então existe uma tarefa i que satisfaça $t(i) = t(j)$ e $\phi(i) > \phi(j)$ tendo k como sucessor. Assim $\phi(i) > \phi(k) > \phi(j)$. A Figura 3.2 ilustra e em 3.3 mostra um exemplo de um grafo e na Figura 3.4 seu escalonamento pelo algoritmo de Hu para $|P| = 3$.

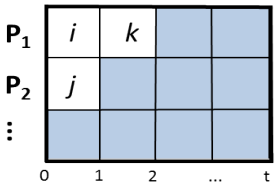

Figura 3.2: Exemplo da alocação de três tarefas com relação de precedência no escalonamento usado pelo algoritmo de Hu.

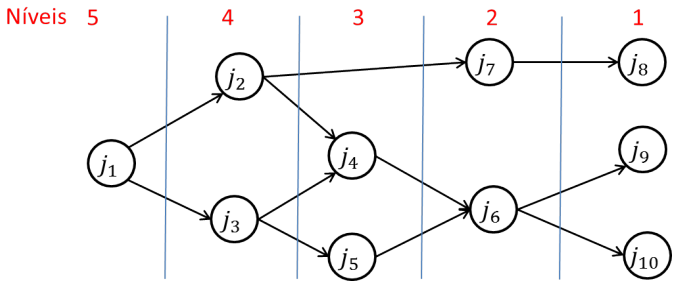

Figura 3.3: Exemplo de um grafo parcialmente ordenado dividido em níveis pelo algoritmo de Hu.

Seja ω_{Hu} o escalonamento gerado pelo algoritmo de Hu e ω_0 o escalonamento ótimo, então para achar um limite superior em $\frac{\omega_{Hu}}{\omega_0}$ é necessário achar um limite inferior para ω_0.

A Figura 3.4 mostra o escalonamento gerado pelo algoritmo de Hu no grafo da Figura 3.3 por meio de uma lista de prioridade e explicando a Figura 3.2. Nesse exemplo j_4 representa o i, j_6 representa k e j_7 representa j.

Seja X_i o conjunto de colunas no escalonamento de Hu e em cada coluna contém i tarefas não vazias, para $i = 1, 2, ..., |P|$.

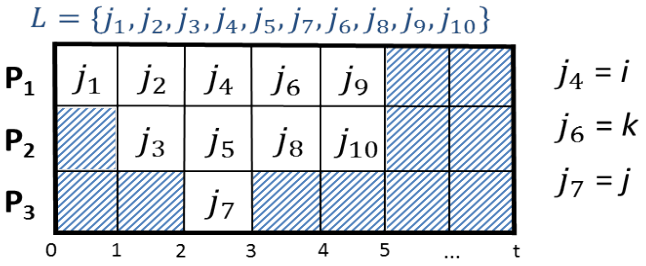

Figura 3.4: Escalonamento do grafo 3.3 com lista de prioridade pelo algoritmo de Hu baseado na explicação da Figura 3.2.

Seja G um grafo parcialmente ordenado em que cada vértice representa uma tarefa e cada arco uma precedência entre duas tarefas que tem tempo de execução unitário, então $X_1 + 2X_2 + ... + |P|X_{|P|}$ tarefas contêm uma cadeia de comprimento no máximo $X_1 + X_2 + ... + X_{|P|}$, assim o comprimento do escalonamento ótimo ω_0 pode ser representado por dois limites inferiores $\frac{X_1+X_2+...+X_{|P|}}{2-\frac{1}{|P|-1}}$ e $\frac{X_1+2X_2+...+|P|X_{|P|}}{|P|}$ para $|P| \geq 3$. Portanto

$$\omega_0 \geq \frac{X_1 + X_2 + ... + X_{|P|}}{2 - \frac{1}{|P|-1}};$$

$$\omega_0 \geq \frac{X_1 + 2X_2 + ... + |P|X_{|P|}}{|P|} (limite\ de\ carga).$$

Assim há dois casos que se segue:

Caso 1: $|P|X_{|P|} \leq X_1 + 2X_2 \ \ \forall \ \ X_{|P|} \geq 0$ (quantidade de tarefas das colunas parciais maior ou igual as completas).

Desde que o escalonamento contenha pelo menos uma coluna parcial o ótimo será $\omega_0 \geq X_1 + X_2 + ... + X_{|P|-1}$ (limite de latência), então

$$\frac{X_1 + X_2 + ... + X_{|P|-1} + X_{|P|}}{X_1 + X_2 + ... + X_{|P|-1}} \leq \frac{\omega_{Hu}}{\omega_0} \leq \frac{2|P| - 3}{|P| - 1} = 2 - \frac{1}{|P| - 1}.$$

Caso 2: $|P|X_{|P|} > X_1 + 2X_2 \ \ \forall \ \ X_1 + X_2 \geq 0$ (quantidade de tarefas das colunas completas maior que parciais).

Assim $|P|X_{|P|} - (X_1 + 2X_2) > 0$. No entanto existe $X_1 + 2X_2 + ... + |P|X_{|P|}$ tarefas no grafo G, consequentemente pelo limite de carga

$$\omega_0 \geq \frac{X_1 + 2X_2 + ... + |P|X_{|P|}}{|P|}.$$

Portanto

$$\frac{X_1+X_2+\ldots+X_{|P|-1}+X_{|P|}}{\frac{X_1+2X_2+\ldots+|P|X_{|P|}}{|P|}} \leq \frac{\omega_{Hu}}{\omega_0} \leq \frac{2|P|-3}{|P|-1} = 2-\frac{1}{|P|-1}.$$

□

Um exemplo de grafo para o caso 1 é a Figura 3.3 que escalonado (Figura3.4) pode ser calculado da seguinte forma.

Existem 4 colunas parciais contendo um total de 7 vértices que representam tarefas e uma coluna completa com três tarefas para $|P| = 3$. Portanto $|P|X_{|P|} = 3$ e $X_1 + X_2 = 7$ (caso 1) $|P|X_{|P|} \leq X_1 + 2X_2$.

$$\frac{X_1+X_2+\ldots+X_{|P|-1}+X_{|P|}}{X_1+X_2+\ldots+X_{|P|-1}} = \frac{5}{4}.$$

$$2-\frac{1}{3-1} = \frac{3}{2}.$$

Assim

$$\frac{5}{4} < \frac{3}{2}.$$

Para que dê a igualdade em ambos valores da esquerda e direita da inequação basta que a quantidade de vértices ou tarefas sejam a mesma nas colunas completas e parciais.

Em caso de haver somente colunas parciais $(X_1, X_2, \ldots, X_{|P|-1})$ o resultado do escalonamento alcançará no máximo 1 + todas as colunas parciais (limite de latência), portanto, o ótimo.

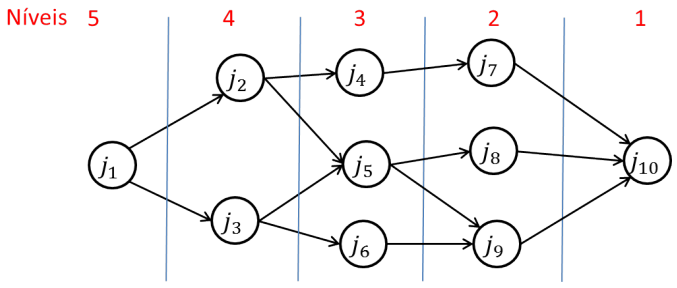

Figura 3.5: Exemplo de um grafo para o caso 2 da prova de aproximação do algoritmo de Hu.

Para o caso 2 ainda para $|P| = 3$ a Figura 3.6 mostra um escalonamento gerado pelo algoritmo de Hu que existem 5 colunas entre elas 3 parciais e 2 completas, totalizando 6

tarefas nas completas e 4 nas parciais. O grafo desse escalonamento encontra-se na Figura 3.5.

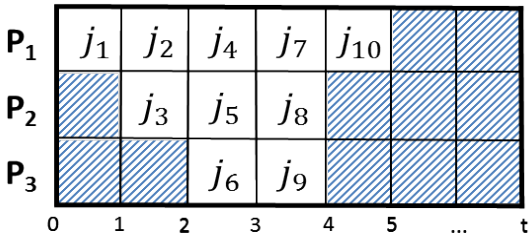

Figura 3.6: Exemplo de um escalonamento gerado pelo algoritmo de Hu no grafo da Figura 3.5 para o caso 2 da prova de aproximação.

Consequentemente

$$\frac{X_1 + X_2 + ... + X_{|P|-1} + X_{|P|}}{\frac{X_1 + 2X_2 + ... + |P|X_{|P|}}{|P|}} = \frac{5}{\frac{10}{3}} = \frac{3}{2}.$$

Com aproximação para $|P| = 3$

$$2 - \frac{1}{3-1} = \frac{3}{2}.$$

Portanto

$$\frac{3}{2} = \frac{3}{2}.$$

Se houvesse duas tarefas k e j a mais para ser executada no tempo de j_{10}, ou seja, $t(k) = t(j) = t(j_{10})$ ainda assim estaria incluso no caso 2, no entanto se uma das tarefas fosse escalonada em $t(j_{10}) + 1$ seria iniciado um novo bloco e esta pertenceria ao caso 1. Se a nova coluna fosse completa continuaria pelo cálculo do caso 2.

3.2 Algoritmo aproximativo de Coffman & Graham (CG)

Primeiramente Graham em 1966 propôs um algoritmo para o problema $P||C_{max}$ e possui caráter histórico, pois foi o primeiro algoritmo conhecido na área de escalonamento a ter uma prova de sua razão de aproximação $2 - \frac{1}{|P|}$, onde $|P|$ é o número de processadores.

[Coffman Jr and Graham(1972)] propuseram um algoritmo para $P|prec, p_j = 1|C_{max}$

que ficou conhecido como "algoritmo de CG" neste trabalho. Eles mostraram que esse algoritmo é ótimo para o caso em que $|P| = 2$. [Lam and Sethi.(1977)] provaram que a razão de aproximação do algoritmo de CG quando $|P| \geq 3$ é de $2 - \frac{2}{|P|}$, onde $|P|$ é o número de processadores paralelos. [Braschi and Trystram(1994)] corrigiram um erro na análise de Lam & Sethi e deram um refinamento no limite inferior, mas que não alterou sua razão de aproximação.

Algoritmo 2: Algoritmo - CG

início

Escolha uma tarefa arbitrária (J_0), tal que (J_0) não tenha sucessores e atribua rótulo 1, isto é, $\alpha(J_0) = 1$;

para cada $i = 2$ *até* n **faça**

a - Seja R o conjunto das tarefas cujo sucessores imediatos já foram rotulados;

b - Seja $j^*, j \in R$ *tal que* $N(j^*) < N(j)$ *(lexicograficamente);*

c - $\alpha(J^*) \leftarrow i$ *(atribuindo um rótulo);*

fim para cada

Seja $L = \{j^k$ e $j^{k-1}\}$, tal que $\alpha(j^k) > \alpha(j^{k-1})$ para $i \leq k \leq n$. (A lista L é o conjunto de tarefas em ordem não crescente dos rótulos);

Execute estratégia de escalonamento na lista de L.

fim

O algoritmo funciona da seguinte forma: primeiro ele atribui um rótulo a uma tarefa que não tenha sucessores e verifica as tarefas que são paralelas a essa tarefa, ou seja, as tarefas que estão no mesmo nível, verifica também as suas tarefas predecessoras imediatas e atualiza o conjunto R das tarefas cujos sucessores não foram rotulados. Nesse conjunto R terá dois subconjuntos $N(j^*)$ e $N(j)$, onde (j^*) é a tarefa que está sendo rotulada e $N(j)$ são as maiores lexicograficamente e, em cada iteração a tarefa (j^*) é rotulada e a próxima tarefa (j) é nomeada (j^*). Por fim será construída uma lista em ordem não crescente dos rótulos e escalonado de acordo com essa lista.

Então, pode-se dizer que o algoritmo de CG é um refinamento do algoritmo de Hu através da introdução de uma estratégia inteligente para escalonar as tarefas com o mesmo nível utilizando uma lista de rótulos e todos os tempos de escalonamento do algoritmo de

CG poderiam ser também produzidos pelo algoritmo de Hu. Portanto, todas as propriedades para o algoritmo de Hu discutidas acima também podem se aplicar ao algoritmo de CG.

Utilizando uma estrutura de dados adequada, o algoritmo de CG pode ser implementado para ser executado com o mesmo tempo de Hu. Se um processador estiver ocioso em algum momento no tempo, pode-se dizer que ele está executando uma tarefa com rótulo zero.

Agora, será mostrado resumidamente a prova da razão de aproximação para o algoritmo de CG, portanto, a prova se apoia nesse algoritmo baseado em lista para o problema $P|prec, p_j = 1|C_{max}$.

Primeiramente será mostrado um exemplo de um sistema de escalonamento com restrição de precedência e tempo unitário tendo uma instância arbitrária com 9 tarefas a serem escalonadas em três processadores ou máquinas paralelas e cada tarefa com tempo unitário de processamento, sendo possível escalonar apenas uma vez em alguma máquina cada tarefa.

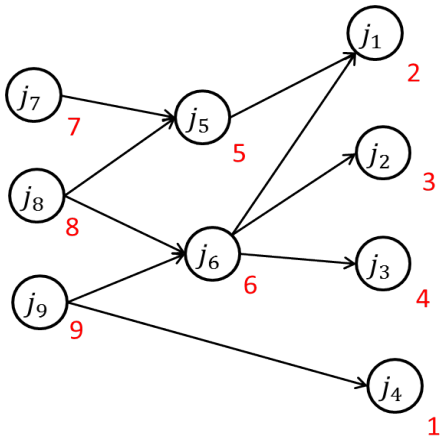

Figura 3.7: Exemplo de um grafo rotulado pelo algoritmo de CG.

Sendo a entrada $([j_9, j_8, j_7, j_6, j_5, j_4, j_3, j_2, j_1], 3)$, onde o segundo parâmetro (3) é a quantidade $|P|$ de processadores representado pelo grafo da Figura 3.7, onde cada tarefa está rotulada (de vermelho) pelo algoritmo. O problema torna-se $P_3|prec, p_j = 1|C_{max}$.

Supondo que o escalonamento gerado pelo algoritmo de CG a partir desses dados encontra-se na Figura 3.8. Esse escalonamento acontecerá se o rótulo 1 ficar no vértice ou tarefa j_4.

Observando-se a contagem na linha do tempo da esquerda para a direita, cada célula

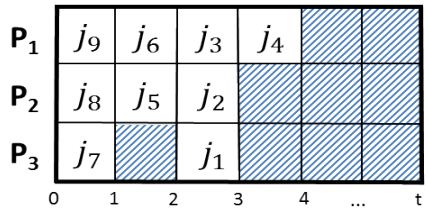

Figura 3.8: Exemplo de escalonamento válido com 9 tarefas e 3 processadores usando o algoritmo de CG no grafo da Figura 3.7.

na horizontal representa o tempo de processamento de um escalonamento possível gerado pelo algoritmo de CG que término no instante igual a 4 a partir do inicio de execução da primeira tarefa e assim por diante até que a última tarefa termine de ser executada no último processador no último instante de tempo. Contudo a melhor possibilidade possível no mesmo grafo da Figura 3.7, é o somatório do tempo de processamento das tarefas dividido pela quantidade de processadores que nos apresenta um escalonamento de tempo ótimo igual a 3 sendo um limite inferior também, haja visto que cada tarefa tem tempo unitário e que só pode ser processada uma única vez como mostra a Figura 3.9. Esse escalonamento acontecerá se o rótulo 1 pertencer a alguma das tarefas j_1, j_2 ou j_3 no exemplo do grafo da Figura 3.7.

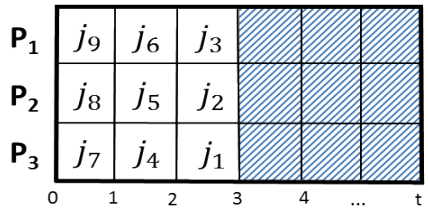

Figura 3.9: Exemplo de um escalonamento ótimo pelo algoritmo de CG obtendo um limite inferior.

Nesse caso do exemplo para essa instância, pode-se calcular dividindo o resultado do tempo de escalonamento gerado pelo algoritmo de CG pelo tempo de escalonamento ótimo e assim surge a razão $\frac{4}{3}$, onde 4 é o resultado do escalonamento gerado e 3 é o tempo do melhor escalonamento possível, ou seja, o ótimo para essa instância.

Mas como fazer essa aproximação de maneira geral para todas as instâncias levando em conta também o número de processadores que serão utilizadas no processo como um todo? dessa maneira para $|P| = 2$ o algoritmo de CG mostra solução ótima, porém para $|P| \geq 3$ não garante o ótimo, contudo apresenta uma razão de aproximação que será explicado a seguir.

Teorema 2: O algoritmo de CG tem razão $2 - \frac{2}{|P|}$ de aproximação para o problema

$P|prec, p_j = 1|C_{max}$ quando $|P| \geq 3$.

Prova: O algoritmo de CG produzirá uma lista L de $1,...,n$ tarefas j para serem alocadas em processadores $P_1, P_2, ..., P_m$ em que $m = |P|$, ou seja, um escalonamento viável se todas as tarefas forem escalonadas.

Seja ω_{CG} o escalonamento gerado pelo algoritmo de CG para uma determinada instância e ω_0 o escalonamento ótimo para essa mesma instância, então:

$$\frac{\omega}{\omega_0} \leq 2 - \frac{2}{|P|}. \tag{3.1}$$

Seja $j \in J$ uma tarefa tal que $\alpha(J)$ constituti um rótulo para cada tarefa $j \in J$, e $t(J)$ é uma unidade de tempo no qual cada tarefa j será alocada no escalonamento.

No escalonamento gerado pelo algoritmo de CG, se a tarefa j é escalonada pelo processador P_1, então j^* é a tarefa escalonada depois de j, assim $t(j) \leq t(j^*)$ isso implica que $\alpha(j) > \alpha(j^*)$ como mostra a Figura 3.10.

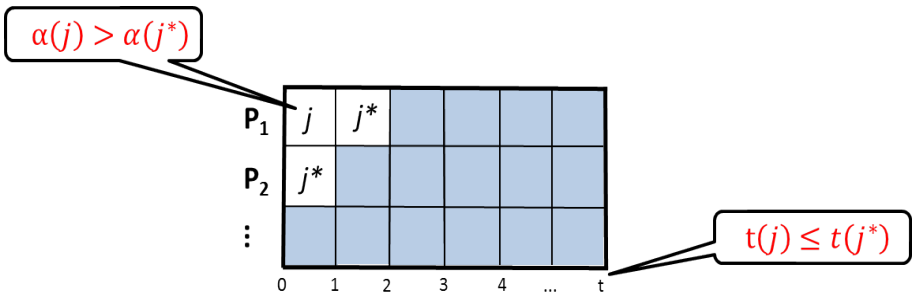

Figura 3.10: Exemplo da alocação de duas tarefas com relação de precedência rotuladas dentro do escalonamento usado pelo algoritmo de CG.

Isso se dá por conta do rótulo de j ser maior que o rótulo de j^*.

Seja S o escalonamento de CG que dividi-se em blocos $S = X_q, ..., X_0$, onde $q \geq 0$. Formando segmentos do tipo $W_0, ...W_r$ para todo $r \geq 0$;

Assumindo que o sentido dos arcos do grafo é da esquerda para a direita então: seja W um segmento a esquerda de W^*, então para todas as tarefas em J que pertencem ao seguimento W e J^* em W^*, J deve ser completado antes de J^*, isto é, $J \prec J^*$. Um exemplo é mostrado na Figura 3.11.

Seja ω_{CG} o tamanho do escalonamento gerado pelo algoritmo de CG no segmento W e ω_0 o tamanho do escalonamento ótimo para as tarefas desse segmento. Então $\frac{\omega_{CG}}{\omega_0} \leq 2 - \frac{2}{P}$ para $|P| \geq 3$, onde $|P|$ é o número de processadores.

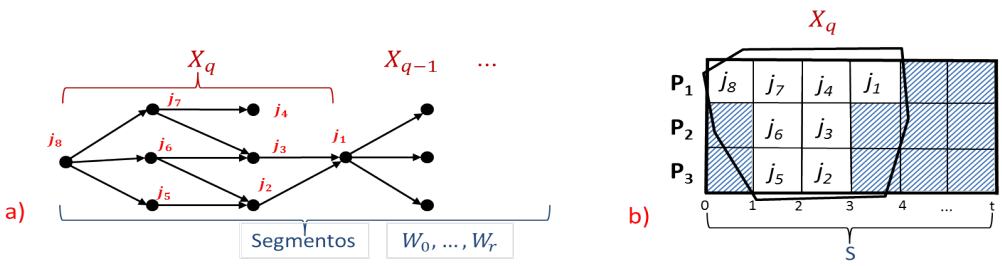

Figura 3.11: Exemplo da alocação em que bloco e segmento com pares de tarefas tem relação de precedências rotuladas dentro do escalonamento.

Seja $X_i, \ldots, X_{i-k}$, onde $k \geq 0$ um bloco no segmento W, e b o número de colunas parciais nestes blocos. Se X_i inicia com uma coluna completa, então o escalonamento das tarefas em W deve ter pelo menos o tamanho $\omega_0 \geq 1+b$ (limite de latência). Como mostra a Figura 3.12.

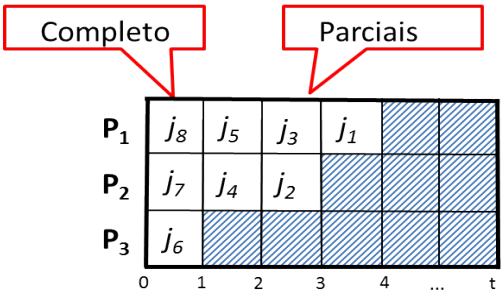

Figura 3.12: Exemplo de escalonamento de um bloco no segmento W com a primeira coluna completa e o restante parcial.

Seja x o número total de tarefas em W, então $x \geq |P|c + 2b - 1$.

$|P|$ é o número de processadores;

c é o número de colunas completas;

b é o número de colunas parciais.

Assim para qualquer escalonamento existem dois casos possíveis, onde o primeiro caso será quando começa com a primeira coluna completa e o segundo quando a primeira coluna for parcial.

Caso 1: se o bloco que estiver mais a esquerda de X_i inicia com coluna completa.

$\omega_0 \geq 1+b$ Assim,

$$|P|\omega_0 \geq x \geq |P|c + 2b - 1.$$

Sabendo que o escalonamento de CG é $\omega_{CG} = (b+c)$, então: multiplicando por $|P|$ de um lado e de outro da equação surge $|P|\omega_{CG} = |P|(b+c)$.

Somando $2b+|P|+3$ de um lado e outro terá

$|P|\omega_{CG}+2b+|P|+3 = |P|c+|P|b+2b+|P|+3$ isolando $|P|\omega$ fica,

$|P|\omega_{CG} = |P|c+|P|b+2b+|P|+3-2b-|P|-3$ modelando matematicamente

$|P|\omega_{CG} = |P|c+2b-1+(|P|-2)(b+1)-|P|+3.$

Substituindo pelo escalonamento ótimo encontra-se

$$|P|\omega_{CG} \leq |P|\omega_0+(|P|-2)\omega_0-(|P|-3).$$

Desde que $|P| \geq 3$, pelos rudimentos matemáticos segue a inequação $\frac{\omega_{CG}}{\omega_0} \leq 2-\frac{2}{|P|}$ conforme demonstrado em (3.1).

Caso 2: se o bloco mais a esquerda de X_i inicia com coluna parcial, significa dizer que as tarefas que estão em X_i precede todas as outras tarefas do segmento. Se houver alguma tarefa extra que tenha o rotulo menor que as outras em X_i ela será pelo menos sucessora de alguma tarefa na coluna parcial que inicia o bloco, então

$$|P|\omega_0 \geq x+1 \geq |P|c+2b.$$

Assim o segmento que existe deve ser uma cadeia de pelo menos $|P|$ tarefas em W, por isso $\omega_0 \geq b$.

Desde que $\omega_{CG} \geq b+c$ pode-se obter

$$|P|\omega_{CG} = |P|(c+b) = |P|c+2b+(|P|-2)b.$$

Substituindo pelo escalonamento ótimo encontra-se

$$|P|\omega_{CG} \leq |P|\omega_0+(|P|-2)\omega_0.$$

Desde que $|P| \geq 3$, segue a inequação $\frac{\omega_{CG}}{\omega_0} \leq 2-\frac{2}{|P|}$ conforme demonstrado em (3.1).

□

3.3 Algoritmo de Garey & Jonhson (GJ)

Esta seção descreve o algoritmo genérico proposto por [Garey and Johnson(1976)] que garante a otimalidade da solução para $|P| = 2$. Neste trabalho será abordado uma síntese

desse algoritmo, pois foi a partir deste que surgiu a proposta de prazo das tarefas proposta por [Gangal and Ranade.(2008)] que será explicado mais a frente.

Algoritmo 3: Algoritmo de Garey e Jonhson (GJ)

início

1 - Calcule os prazos de todas as tarefas respeitando precedências;

2 - Escolha qualquer tarefa disponível;

3 - Escalone no primeiro processador livre;

3.1 Escalone primeiro os antecessores com prazo menor.

4 - Repita até que todas as tarefas estejam escalonadas.

fim

Será feita uma sucinta dedução para achar a 2-aproximação deste algoritmo.

A alocação das tarefas nos processadores $|P|$ somente terá duas situações:

- Coluna completa: todos processadores ocupados.
- Coluna parcial: algum processador está ocioso.

Para o caso de serem somente colunas parciais o tamanho do escalonamento pode ser no máximo o comprimento do caminho mais longo (H), então é um limite inferior.

O prazo máximo - o prazo minimo + 1 é o escalonamento gerado pelo algoritmo de GJ (limite de latência).

Primeiro limite: prazo máximo - o prazo mínimo + $1 \geq H$, portanto $\omega_0 \geq H$.

Para o caso de terem somente colunas completas, então $\lceil \frac{N}{|P|} \rceil$ (limite de carga) é o segundo limite gerado, portanto, prazo máximo - o prazo minimo + $1 \geq \lceil \frac{N}{|P|} \rceil$ e assim $\omega_0 \geq \frac{N}{|P|}$.

Mas um escalonamento é formado por colunas parciais e completas, assim o prazo será, prazo $\lceil \frac{N}{|P|} \rceil + H = OPT + OPT = 2OPT$. Que melhorado pode chegar a $2 - \frac{1}{|P|}$.

3.4 Algoritmo aproximativo de Gangal & Ranade (GR)

Esse algoritmo foi proposto por Devdatta Gangal & Abhiram Ranade em 2008 [Gangal and Ranade.(2008)] e provada sua razão de aproximação $2 - \frac{7}{3|P|+1}$, onde $|P|$ é a

quantidade de processadores para o caso de $|P| > 3$, portanto, tendo melhor aproximação que os anteriores.

Algoritmo 4: Algoritmo - (Gangal & Ranade)

Entrada: Grafo $G(V,E)$, $|P|$: Conjunto de processadores.
início
  1 - Inclua um vértice a com arestas $\forall v \in V(G)$
  para cada *vértice $v \in V(G)$* **faça**
    se *v é vértice sem sucessores* **então**
      Seja $D(v) = \Delta$, onde Δ é um número qualquer, $D(v)$ = comprimento do caminho mais longo em G;
    senão
      Calcule os prazos $D(v)$ para os outros vértices de acordo Equação (3.2);
    fim se
  fim para cada
  2 - Ordene os vértices em ordem não decrescente de acordo com seu prazo, fazendo a redução de transitividade ;
  3 - Escalone a no tempo 1 *do processador* 1*;*
  para cada *vértice $v \in V(G)$* **faça**
    enquanto *Reorganiza predecessores (v)* **faça**
      1. Seja t' uma coluna com maior prazo, tal que contém alguns predecessores de v;
      2. Seja A o conjunto de vértices na coluna de $t' - 1$ e coluna t';
      se *todos os vértices em A tiver o mesmo prazo, e A tem no máximo $|P|$ predecessores de v, e coluna t' tem menos de $|P|$ vértices* **então**
        - Mova os predecessores de v para a posição $t' - 1$;
        - Mova outros vértices da coluna $t' - 1$ para a coluna t' se for necessário;
      fim se
    fim enqto
    Seja t o menor tempo em que v pode ser escalonado;
    Escalona v na coluna t livre do primeiro processador disponível.
  fim para cada
fim

Para simplificação será referenciado ao algoritmo de Gangal & Ranade somente com as iniciais dos nomes como foi feito com o algoritmo de CG, portanto será chamado algoritmo de GR, ou "algoritmo (3)".

A Equação 3.2 calculará todos os prazos dos vértices v assim como dará suporte na explicação desse algoritmo. (Mais detalhes esclarecidos em [Lever and Latorre(2016)]).

Cada vértice folha será calculado pelo comprimento do caminho mais longo no grafo G. N é a quantidade de sucessores de v no caminho mais longo até suas folhas. L é o número de arcos desse caminho (v,u), $|P|$ são processadores paralelos e D o maior prazo

do caminho de v até u.

$$D(v) = min_{L,D}\{D - L - \lceil \frac{|N(v,D,L)|}{|P|} \rceil\}. \tag{3.2}$$

Esta equação é utilizada para calcular os prazos $D(v)$ para cada vértice v no sentido inverso da ordem de classificação começando pelas folhas em que $N(v,D,L)$ é o maior conjunto de vértices (v,u) e será calculado o menor prazo para v, tendo um caminho de comprimento, pelo menos, $L+1$ arco que liga v até u e possuindo $D(u) \leq D$, onde D é o prazo do caminho mais longo no grafo (folhas). Esta execução pode ser feita em tempo polinomial. O máximo alcance de valores inteiros que $L, D, |N(v,D,L)|$ pode tomar é $|G|$, contudo a complexidade de tempo do algoritmo é $|G|^4$ para calcular todos prazos. A Figura 3.13 mostra um exemplo de um grafo com todos os prazos calculados de acordo com a Equação (3.2).

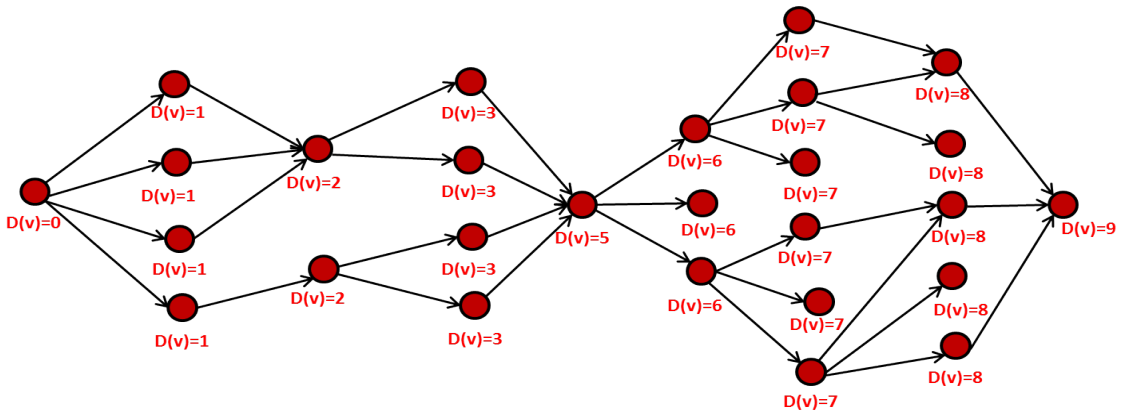

Figura 3.13: Exemplo de um grafo com todos os prazos dos vértices calculados para 4 processadores pelo algoritmo de GR com a Equação 3.2 e feito a redução de transitividade.

O funcionamento do algoritmo de GR consiste primeiramente em criar um vértice artificial "a" ligando a todos os outros vértices, para estimar prazos $D(v)$ para todos os vértices. Definindo $D(v)$ um prazo para todos os vértices folha da instância como o comprimento do caminho mais longo. Os caminhos dos outros vértices são então calculados utilizando a Equação 3.2, mas para isso fixa a quantidade de processadores $|P|$ como instância junto com o grafo que se quer usar, pois se faz necessário para o cálculo da Equação 3.2. Após ser calculado os prazos o algoritmo organiza o grafo fazendo a redução de transitividade.

No momento em que os vértices estiverem sendo escalonados será feito em ordem não decrescente do prazo, começando com "a" no tempo 1 no processador 1 que é o menor

prazo existente. Todos os prazos $D(v)$ são consistentes com precedência, ou seja, para qualquer arco (v,u), tem $D(v) < D(u)$. Assim, terá certeza de que "a" tem o menor prazo e que os antecessores de cada vértice v no segundo "*Para*" do algoritmo principal já estão em posições de serem escalonados. Dentro desse "*Para*" o algoritmo irá reorganizar os antecessores de v, se possível, essa possibilidade vai depender da precedência e da quantidade de vértices críticos existentes.

Em um grafo G serão chamados de vértices críticos v aqueles que são precedentes a um vértice u no caminho (v,u).

Seja $v \prec u$, então v é critico a u, assim, $D(v) < D(u) \forall (v,u)$ em G.

No passo da reorganização dos predecessores os vértices críticos que estão na coluna $t'-1$ e $t' \in A$ e A tendo no máximo $|P|$ vértices críticos a v terá remanejamento de algum vértice que não seja crítico indo da coluna $t'-1$ para t' e, assim os vértices da coluna t' indo para $t'-1$. Se todos os antecessores de v se moverem para a coluna de $t'-1$, em seguida, v vai escalonar na coluna t'. O benefício preciso desse rearranjo são apresentados na Figura 3.14.

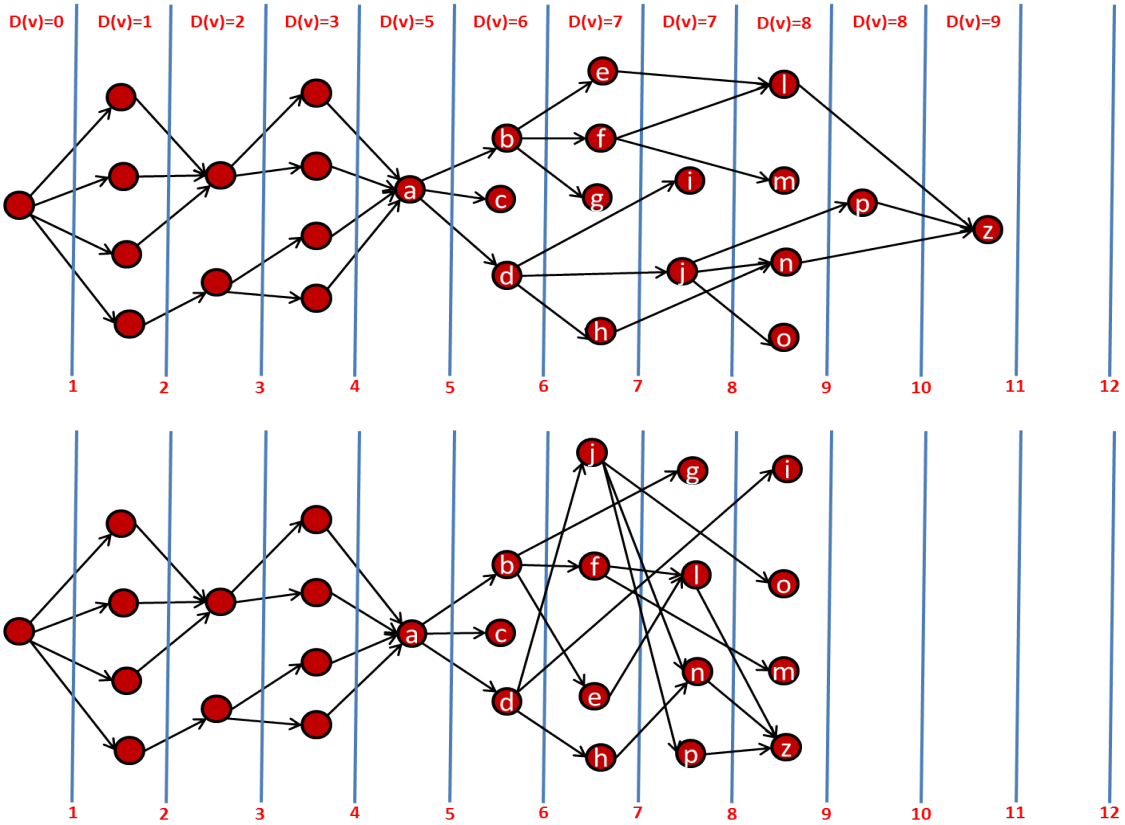

Figura 3.14: Exemplo do grafo sendo escalonado pelo algoritmo de GR com a reorganização dos predecessores para $|P| = 4$.

A Figura 3.14 mostra que a reorganização dos predecessores no primeiro bloco não é possível, no entanto no segundo bloco da instância do exemplo e o tempo é quebrado, isso ocorre devido a reorganização das tarefas seguintes.

A tarefa "j" que iniciava no tempo 7, agora, foi inicia no tempo 6, por consequência a tarefa "g" que estava iniciando no tempo 6 inicia no tempo 7. Com isso as tarefas sucessoras de "j" {n, p, o} e consequentemente "z" diminuem um período de tempo para o inicio da execução.

As tarefas "l" e "n" podem iniciar no tempo 7, pois no tempo 7 a quantidade de tarefas nessa coluna é menor que $|P|$ processadores.

A tarefa "i" como é uma folha e tem pouca prioridade em vez de iniciar no tempo 7 inicia no tempo 8.

A tarefa "p" vai para o tempo 7 e consequentemente "z" vai para o tempo 8. Terminando assim o escalonamento.

Após essa reorganização o tempo cai de 11 para 9, mas 9 é o comprimento do caminho mais longo no grafo G e portanto é um limite inferior (limite de latência), assim a Figura 3.14 mostra o escalonamento ótimo para esse exemplo de grafo em 4 processadores.

Agora será mostrada a dedução da prova da razão de aproximação do algoritmo 3 para o caso de $|P| > 3$, onde $|P|$ é o número de processadores.

Teorema 3: o algoritmo de Gangal & Ranade (GR) encontra um escalonamento com razão $2 - \frac{7}{3|P|+1}$ de aproximação do ótimo, onde o número de processadores é $|P| > 3$.

Prova: Será pensado no escalonamento como um vetor de duas dimensões, com coluna t indicando os vértices escalonados no tempo t. Usar-se-á $(t, |P|)$ para denotar o vértice escalonado na coluna do tempo t no processador $|P|$.

Seja S o comprimento do escalonamento de GR. Então existe um conjunto de colunas adjacentes $[v, v+1, ..., u]$ que será chamado de uma região, e será escrito como $[v,u]$, onde $v \prec u$. Seja $d([v,u])$ o prazo inferior de $[v,u]$ definido como um $D(u+1,1) - D(v,1)$. A coluna de v terá uma queda no prazo se $D([v,v]) > 0$. Definido por $L([v,u]) = u - v + 1$ (limite de latência) em caso de ter colunas completas, onde existe um caminho de v para u que contenha $L+1$ arcos e L é a quantidade de arcos ligados de v até u definindo o caminho mais longo em (v,u) do grafo G.

O resultado principal segue a partir de três limites inferiores e será provado em $d([1, S-1])$. O primeiro limite é "*load bound*" (limite de carga) e será chamado $d(X)$,

onde o somatório do processamento do tempo das tarefas são divididas pela quantidade de processadores $|P|$, como já foi dito anteriormente, então vale $\frac{tarefas}{processadores}$. O segundo limite é "*Latency bound*" (limite de latência $L(X)$) o tempo deve ser pelo menos o comprimento do caminho mais longo. Se por acaso todos os *slots* tiverem a quantidade de no máximo $|P|$ vértices, então $L(X) = \omega_0$ e será um limite inferior. E por fim, o terceiro limite que não tem haver com os outros e sim com relação aos *slots* dentro dos blocos quando há quebra de prazo modificado nos *slots* bons fazendo com que o remanejamento de tarefas ajuste os *slots* ruins.

Para que fique caracterizado a essência do algoritmo de prazo modificado o ótimo será chamado $d(X)$ (que é o prazo de uma região) para o limite de carga e $L(X)$ para limite de latência, portanto $d(X) = \omega_0$ e $L(X) = \omega_0$.

Denotando os *slots* dentro do escalonamento $\sum_{i=1}^{|P|} iX_i = 1X_1, 2X_2, ..., |P|X_{|P|}$ será chamado de bloco o conjunto de *slots* desde um vértice inicial v até um vértice de corte u. Uma região é denominada por um intervalo entre v e u e tem o prazo $d([v+1, u-1])$. Assumindo que $d(X) = \omega_0$ e $d(X)$ é uma região, então será usado X_1, X_2, X_g, X_f, X_b para denotar o número de *slots* nessa região. Onde:

X_1 é o *slot* com um único vértice crítico;

X_2 é o *slot* com dois vértices críticos;

X_g será chamado de *slot* bom aquele que pode ser quebrado e tem pelo menos três vértices críticos;

X_b é o *slot* ruim, o qual não pode ser quebrado.

X_c será chamado de *slot* completo. Um *slot* é completo se ele tem somente $|P|$ vértices críticos, caso contrário, é parcial.

Seja $d(X)$ o prazo de uma região no bloco X como sendo o limite inferior desse bloco e $L(X)$ o caminho mais longo do escalonamento no mesmo bloco. Se todos os *slots* no bloco X forem parciais, então $\frac{L(X)}{d(X)} \leq 2 - \frac{7}{3|P|+1}$, para $|P| > 3$.

Pelo primeiro limite é conhecido que

$$d(X) \geq \frac{X_1 + 2X_2 + 3X_g + |P|X_c + |P|X_b}{|P|}. \tag{3.3}$$

Pelo segundo limite obtêm-se

$$d(X) \geq 1 + X_1 + X_2 + X_g. \tag{3.4}$$

Pelo terceiro limite terá

$$d(X) \geq 2 + X_2 + \frac{X_b - 1}{3}. \tag{3.5}$$

A função objetivo é

$$max \frac{L(X)}{d(X)} = \frac{2 + X_2 + X_g + X_b}{d(X)}. \tag{3.6}$$

Portanto, pode-se reescrever as desigualdades substituindo os seguintes valores $Q = d(X) - 2$ e $Y_b = X_b - 1$ nas equações (3.3), (3.4), (3.5) Desse modo fica

$X_2 + X_g \leq Q.$

$Y_b + \frac{1+2X_2+3X_g}{|P|} \leq Q.$

$X_2 + \frac{Y_b}{3} \leq Q.$

Dessa forma a função objetivo pode ser escrita como

$max \frac{3+X_2+X_g+Y_b}{2+Q}.$

Assim todas as desigualdades a cima devem ser colocadas como ótimo em

$X_2 = \frac{2|P|Q+3Q+1}{3|P|+1}.$

$X_g = \frac{|P|Q-2Q-1}{3|P|+1}.$

$Y_b = \frac{3|P|Q-6Q-3}{3|P|+1}.$

Partindo desse valor a função objetivo passa a ser vista como sendo:

$2 - \frac{7Q+4+3|P|}{(3|P|+1)Q+6|P|+2}.$

Que é no máximo $2 - \frac{7}{3|P|+1}$ para $|P| \geq 4$, e no máximo $\frac{4}{3}$ para $|P| = 3$, sendo $Q \geq 1$. Para $|P| = 2$ foi provado em [Hu(1961)], [Coffman Jr and Graham(1972)], [Garey and Johnson(1976)] e provado as aproximações em [Chen(1975)] e [Lam and Sethi.(1977)].

□

3.5 Resumo do capítulo

Neste capítulo foram reescritos e detalhados quatro algoritmos da literatura, sendo três deles aproximativos no que se refere a $|P| \geq 3$ e um eficiente. Foi reescrito também sucintamente e detalhadamente a prova de dedução da razão de aproximação de três deles.

Os algoritmos da literatura apresentados neste capítulo são: algoritmo de Hu, o qual

usa estrategia de nível e foi escrito de forma simples e detalhado seu funcionamento em determinadas classes de grafo, assim como sua razão de aproximação. Algoritmo de Coffman & Graham (CG), que também utiliza estratégia de nível foi detalhado e mostrado sua razão de aproximação, assim como seu comportamento em determinados grafos. Algoritmo de Garey & Jonhson (GJ), que não está classificado como aproximativo e sim como eficiente, mas que foi mostrado de forma simples que pode ser 2-aproximativo no pior caso. E por fim o algoritmo de Gangal & Ranade (GR), o qual também foi explanado e detalhado juntamente com sua razão de aproximação demostrando que é o melhor aproximativo até o presente momento.

Capítulo 4

Análise teórica comparativa de algoritmos aproximativos para $P|prec, p_j = 1|C_{max}$

Neste capítulo será apresentado uma breve comparação analítica sobre o comportamento dos algoritmos de Hu e GR para determinadas classes especificas de grafo com restrição de precedência (*in-tree* e *out-tree*). Será mostrado também uma simples prova da otimalidade do algoritmo de GR para árvores enraizadas e comparações das razões de aproximação dos algoritmos de Hu, CG e GR assintoticamente analisando quando cresce a quantidade de processadores $|P|$.

4.1 Análise comparativa dos algoritmos de GR e Hu para *in-tree* e *out-tree*

Nesta seção será mostrada uma breve análise do algoritmo de GR para as classes específicas de instâncias *in-tree* e *out-tree* comparando com o algoritmo de Hu.

Cada vértice no grafo representa uma tarefa e cada arco é representado pela restrição de precedência entre as tarefas que são ligadas pelos arcos.

É conhecido na literatura [McHugh(1984)] que o algoritmo de Hu usa a técnica HLF e mostra o ótimo para árvores enraizadas, assim como tem razão de aproximação $2 - \frac{1}{P-1}$ [Chen(1975)] em que $|P| \geq 3$ para grafos arbitrários, como foi explanado no Capítulo 3, assim se for mostrado que o algoritmo de Hu tem bom comportamento no escalonamento

para *int-tree* e *out-tree* juntamente com o algoritmo de GR, isso fará com que ambos tenham bom comportamento para essas mesmas instâncias.

Na Seção 3.4 deste trabalho foi exposto o algoritmo de GR e explicado como é o seu funcionamento, então para esta análise será tirada a função que reorganiza os predecessores e cada nível em que o algoritmo de Hu organiza será como um slot que tem os mesmos prazos calculados por GR, portanto o algoritmo ficaria da seguinte forma:

Algoritmo 5: Algoritmo modificado - (Gangal & Ranade)

Entrada: Grafo $G(V,E)$, $|P|$*: Conjunto de processadores.*
início
  1 - Inclua um vértice a com arestas $\forall v \in V(G)$
  para cada *vértice* $v \in V(G)$ **faça**
    se *v é vértice sem sucessores* **então**
      Seja $D(v) = \Delta$*, onde* Δ *é um número qualquer,* $D(v)$ *= comprimento do caminho mais longo em G;*
    senão
      Calcule os prazos $D(v)$ *para os outros vértices de acordo Equação (3.2);*
    fim se
  fim para cada
  2 - Ordene os vértices em ordem não decrescente de acordo com seu prazo, fazendo a redução de transitividade ;
  3 - Escalone a no tempo 1 *do processador* 1*;*
  para cada *vértice* $v \in V(G)$ **faça**
    Seja t o menor tempo em que v pode ser escalonado;
    Escalona v na coluna t livre do primeiro processador disponível.
  fim para cada
fim

Lema 1: o algoritmo de GR tem o mesmo comportamento no escalonamento que o algoritmo de Hu para *in-tree* e *out-tree*.

Dado o algoritmo de Hu mostrado na Seção 3.1 será analisado primeiro para *in-tree* e comparado essa mesma instância com o algoritmo de GR. Se for mostrado que para *in-tree* ambos algoritmos se comportam da mesma maneira no escalonamento, então para *out-tree* será válido também e explicado posteriormente. A Figura 4.1 é um conjunto parcialmente ordenado (POSet) enraizado da família de árvores que tem a direção dos arcos da folha até a raiz (*in-tree*).

Prova: Se a raiz da árvore é a última tarefa e Δ é $D(v)$ da última tarefa e essa tarefa não tem sucessor imediato, então essa mesma tarefa tem seu prazo o comprimento do caminho mais longo na árvore. Se existirem tarefas predecessoras, então essas tarefas terão prazo

$D(v) \leq \Delta$ e consequentemente até chegar no vértice artificial "a" que terá o menor prazo $D(v)$ nesta árvore como mostra a Figura 4.1. Em *in-tree* a raiz se transforma em folha para ser calculado os prazos e o comprimento do caminho mais longo que equivale a altura da árvore.

Se for utilizado dois processadores, pelo Capítulo 3, é conhecido que o resultado será ótimo em ambos algoritmos, no entanto aqui será analisado para três e quatro processadores somente e a partir dai a ideia é sucessiva.

Lema 2: Para *in-tree* e *out-tree*, todo nível organizado pelo algoritmo de Hu é um slot organizado pelo algoritmo de GR, assim $\forall\ l(Hu) = X_i(GR)$, onde l é o nível e X o slot.

Prova: Os níveis na parte inferior representam a organização feita pelo algoritmo de Hu e na parte superior os prazos sendo representado pelo algoritmo de GR, que nesse caso cada vértice do mesmo nível terá o mesmo prazo levando em conta que será escalonado em quatro processadores.

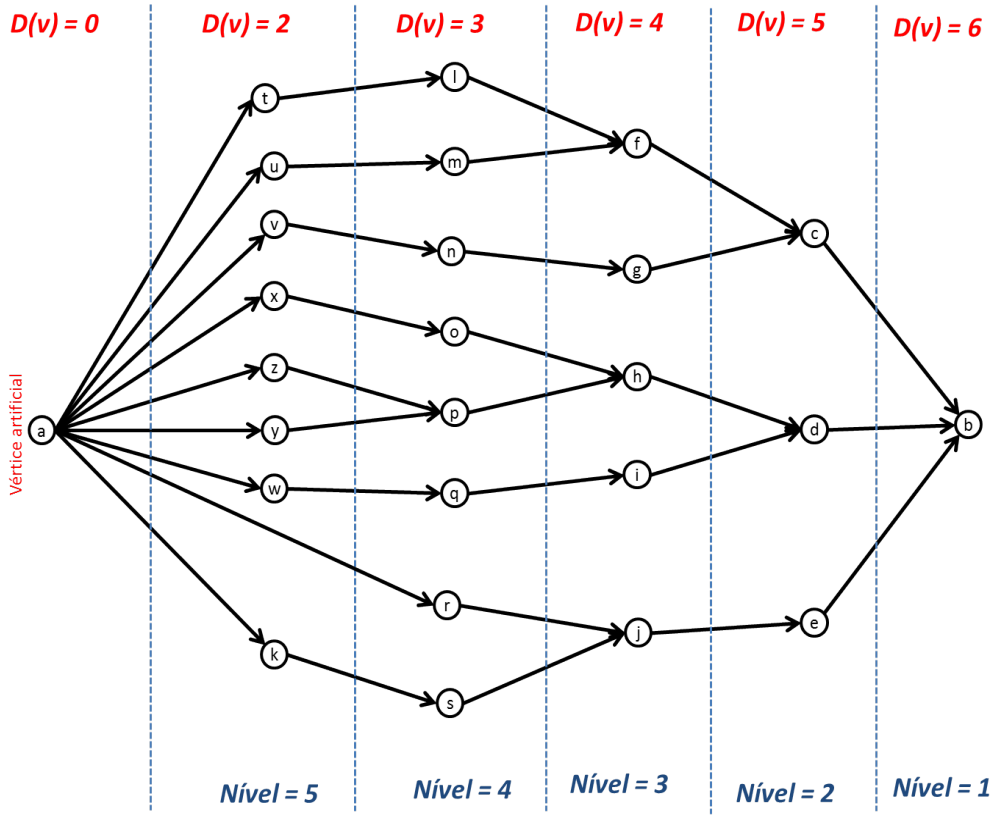

Figura 4.1: Exemplo de uma *in-tree* com redução transitiva organizada por níveis no algoritmo de Hu e por prazos no algoritmo de GR para $|P| = 4$.

É perceptível que enquanto o Algoritmo (5) organiza em $D(v)$ para depois escalonar em ordem crescente desses prazos o algoritmo de Hu organiza de forma crescente dos

níveis para depois escalonar por ordem decrescente desses níveis. Cada slot será representado por um nível. Um exemplo é mostrado na Figura 4.1.

Se houver apenas um vértice em um nível qualquer este mesmo vértice será de articulação ou de corte e iniciará um novo bloco, porém, caso contrário não, portanto todo grafo pode ser um único bloco se e somente se não houver vértice de corte. Todos os vértices que estiverem no mesmo nível terá a mesma distância até a raiz, assim se estiverem os mesmos prazos também terão a mesma distancia até a raiz.

A Figura 4.2 mostra a verificação desse mesmo grafo da Figura 4.1 para 3 processadores. Na mesma *in-tree* a diferença somente se mostra pelo algoritmo de GR até este ponto e isso acontece devido o cálculo dos prazos que se alteram a partir da altura que equivale ao número de processadores.

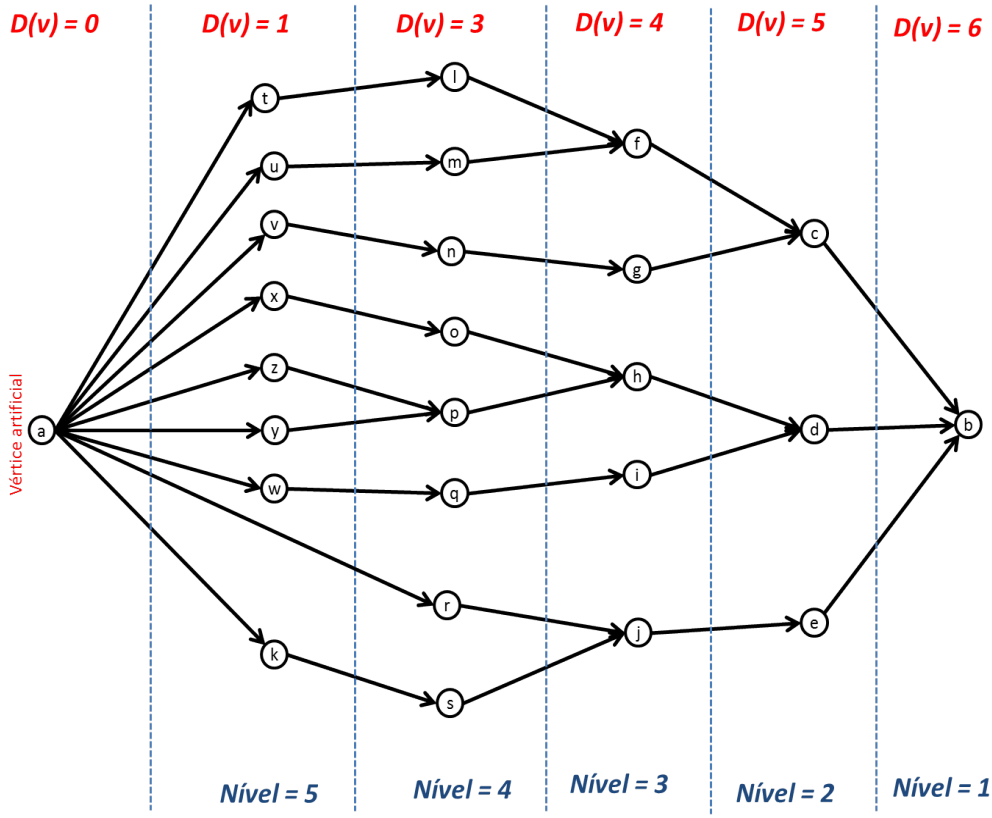

Figura 4.2: Exemplo de uma *in-tree* organizada por níveis no algoritmo de Hu e por prazos no algoritmo de GR para $|P| = 3$.

Para que fique claro este resultado será analisado o comportamento de ambos algoritmos no escalonamento. A Figura 4.3 mostra como pode ser escalonado essa *in-tree* com os algoritmos de Hu e GR (sem a função de reorganizar predecessores).

Com essa observação é possível assumir que ambos algoritmos tem o mesmo com-

portamento no escalonamento e por isso se um é ótimo para essa classe especifica de instância o outro também será.

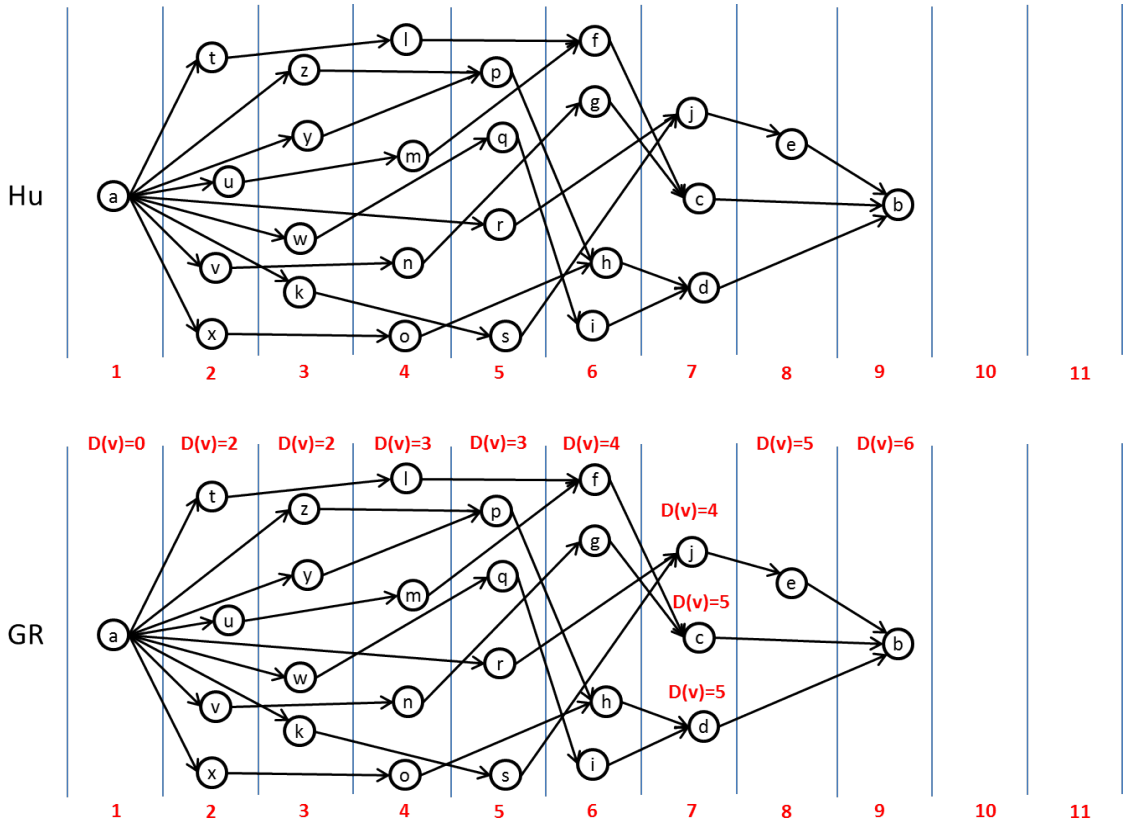

Figura 4.3: Exemplo da organização para escalonar uma *in-tree* nos algoritmos de Hu e GR para $|P| = 4$.

A quantidade de vértices existente nos blocos com prazo $D(v) = 1,3,4$ é múltiplo de $|P|$ e portanto para escalonar todos seria necessário pelo menos mais uma unidade de tempo empurrando os vértices sucessores mais a frente.

No tempo 7 o vértice "j" que tem o prazo 4 foi escalonado juntamente com "c" e "d" que tem o prazo 5, isso porque não existem relações de precedência entre ambos, ou seja, são paralelos e o vértice "e" não pôde preencher o quarto processador porque é sucessor de "j" que está no tempo 7.

Na organização para 3 processadores o comportamento para ambos algoritmos se repete, assim como os procedimentos de prazos diferentes na mesma coluna. A Figura 4.4 mostra esse exemplo.

Ambos algoritmos irão executar no mesmo momento as tarefas equivalentes, mesmo tendo estratégias algorítmicas diferente, mas que no resultado são semelhantes independentemente da quantidade de processadores. Nesse caso específico mesmo que o algoritmo de GR tivesse a reorganização dos predecessores não iria alterar esse tempo de

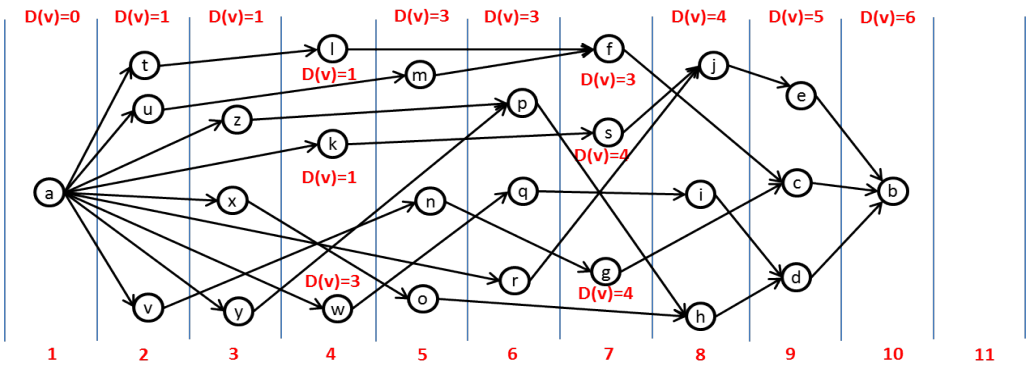

Figura 4.4: Exemplo da organização de uma *in-tree* nos algoritmos de Hu e GR para 3 processadores.

escalonamento, pois não seria possível que o algoritmo executasse essa função.

Assim se o Algoritmo (3) é melhor que o (1) em uma instância arbitrária com precedência, então a melhoria está no que foi retirado do Algoritmo (3) que foi a reorganização dos predecessores e isso fica claro ao comparar ambos algoritmos. Foi levado em consideração a organização da qual as tarefas comportaram-se ou organizaram-se segundo cada algoritmo para ficarem disponíveis aos processadores.

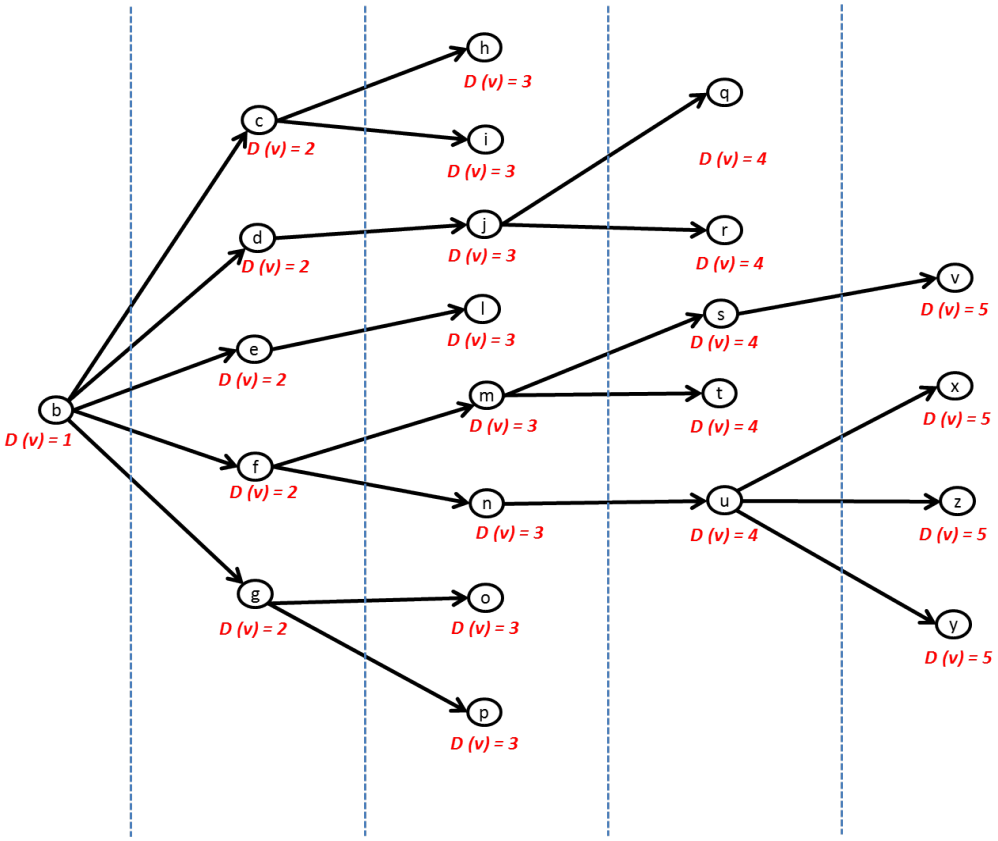

Figura 4.5: Exemplo de uma *out-tree* organizada por prazos no algoritmo de GR para $|P| = 4$.

Considerando as informações já mostradas sobre o caso do grafo *in-tree*, agora será

mostrado o caso quando a instância for *out-tree*. Para essa análise será utilizado o mesmo Algoritmo 4, porém agora além de ser tirado a reorganização dos predecessores também ficará de fora a inclusão de um vértice artificial, pois como a instância é *out-tree* e essa é enraizada com os arcos que sai da raiz e vai em direção as folhas seria desnecessário a inclusão dessa função.

A Figura 4.5 mostra uma *out-tree* típica com prazos calculados pelo algoritmo de GR usando 4 processadores. Para essa família de grafo percebe-se que cada filho tem somente um pai e que todos são descendentes de um único pai que é a raiz.

Usando o mesmo grafo para ser organizado pelo algoritmo de Hu os níveis são mostrados na Figura 4.6. A distância a partir da raiz a qualquer um outro vértice na árvore é a altura do nível entre eles. A Figura 4.7 demostra a organização no escalonamento deste grafo em ambos algoritmos para $|P| \geq 4$.

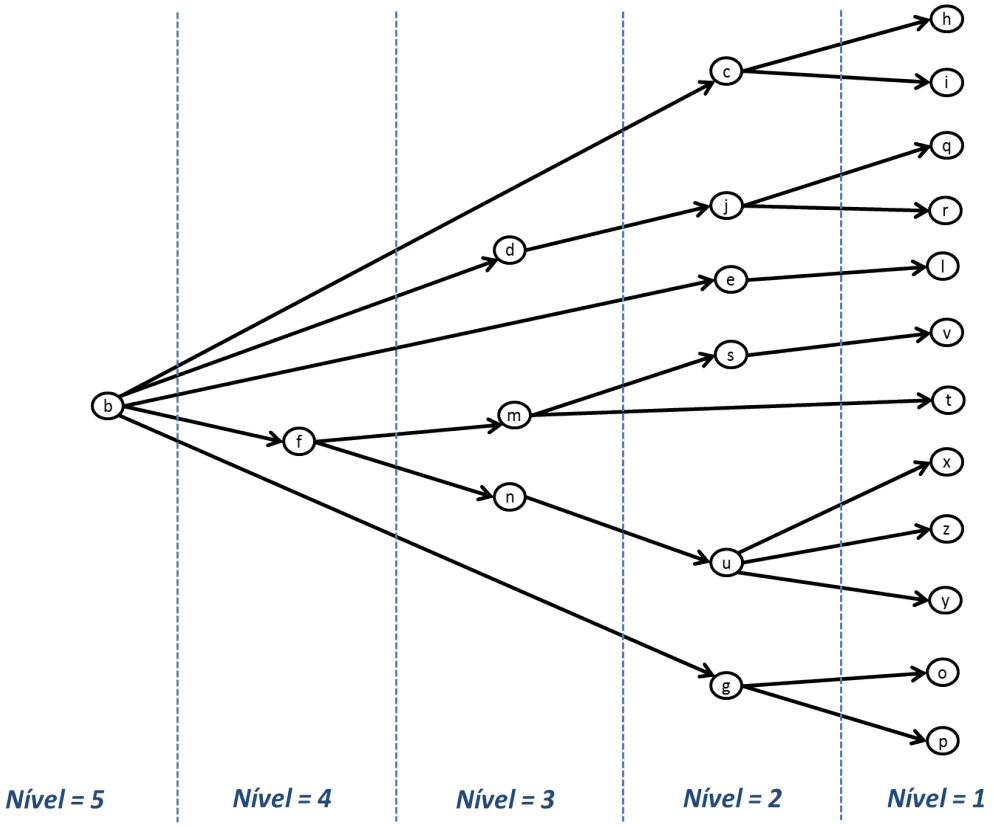

Figura 4.6: Exemplo da Figura 4.5 organizada por níveis no algoritmo de Hu.

A reorganização dos predecessores é possível ter efeito em *out-tree* se houver uma quantidade de folhas com o comprimento da raiz até essa folha menor que o comprimento do caminho mais longo e quando a quantidade de processadores for menor que a largura da *out-tree*. No caso de *in-tree* todos os predecessores são críticos, mas no caso de *out-tree*

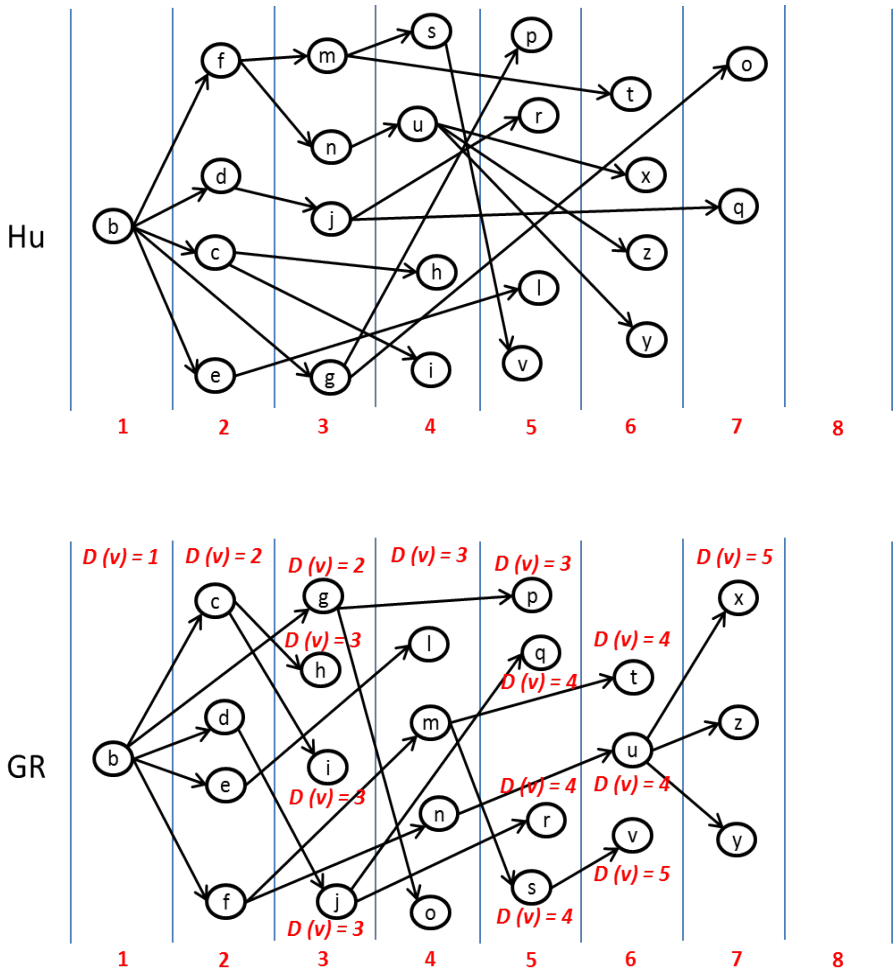

Figura 4.7: Exemplo de uma *out-tree* organizada no escalonamento dos algoritmos de Hu e GR para $|P| = 4$.

não terá essa garantia.

Observando a Figura 4.7 percebe-se que devido a quantidade de vértices que estão no mesmo slot ou nível ser maior que a quantidade de processadores, então algumas colunas no escalonamento de GR serão organizadas por vértices com prazos diferentes, por exemplo no tempo 3 o vértice "g". Isso também acontecerá com o escalonamento de Hu, na qual vértices com níveis diferente estão na mesma coluna de tempo.

Analisando ambos algoritmos organizando esse grafo é possível perceber que há poucas diferenças entre eles e que a organização de nível por nível facilita o escalonamento e se por ventura for retirado dois trechos do algoritmo de Gangal & Ranade (GR) a inclusão do vértice artificial ligados a todos e a reorganização dos predecessores é perceptível que podem ter o mesmo comportamento no escalonamento para *in-tree* e *out-tree*, portanto se o algoritmos de GR tem melhor aproximação que o de Hu é devido a essas funções.

Uma das possibilidades do algoritmo de GR escalonar essa *out-tree* do exemplo

usando a reorganização dos predecessores está na Figura 4.8 que não melhorou o tempo, contudo a organização ficou semelhante a de Hu.

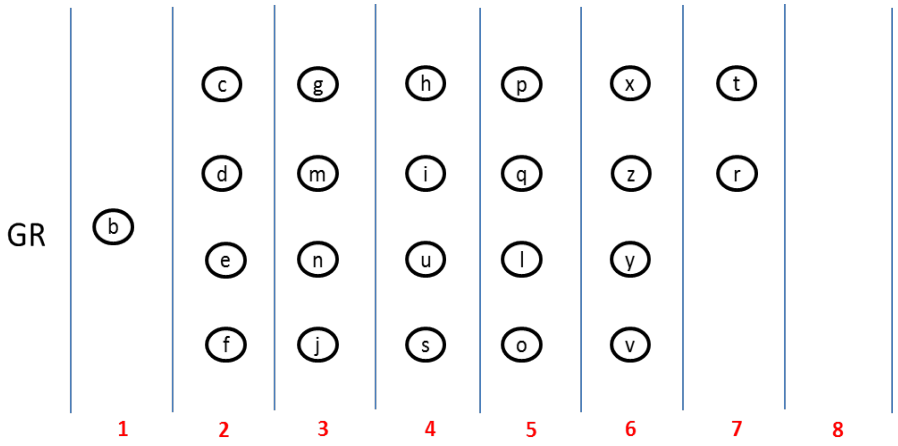

Figura 4.8: Exemplo de uma das possibilidades de organização da *out-tree* no Algoritmo 3 para $|P| = 4$.

□

□

Gangal & Ranade propuseram o algoritmo em 2008 e tiveram a oportunidade de tirar um pouco de proveito da ideia dos anteriores como por exemplo de Hu enraizar as DAGs e organizar por níveis, de Garey & Johnson calcular os prazos e etc. Assim fica claro que para *in-tree* ambos algoritmos se comportam da mesma forma, porém para *out-tree* há possibilidade de convergência. A melhoria existente na aproximação do Algoritmo 3 em relação ao 1 fica claro que não está em grafos do tipo *in-tree* e *out-tree*.

4.2 Prova de otimalidade do algoritmo de GR para árvores de precedência (arborescências)

Nesta seção será mostrada uma simples prova do algoritmo de GR para árvores de precedência no que se refere ao problema $P|prec, p_j = 1|C_{max}$ como principal contribuição desta pesquisa.

Um conjunto finito de tarefas que devem ser escalonadas em um conjunto $|P|$ de processadores ou máquinas. Cada tarefa requer uma unidade de tempo para concluir $p_j = 1$. O escalonamento é limitado por restrições de precedência que existem entre as tarefas. As restrições de precedência são modeladas por uma árvore enraizada. Os vértices da árvore

correspondem às tarefas, e os arcos as precedências entre essas tarefas. O objetivo é escalonar todas as tarefas de modo a minimizar o tempo total de conclusão do escalonamento.

O algoritmo que Gangal & Ranade propuseram para resolver este problema está explicado de forma simples no Capítulo 3 deste trabalho.

Teorema 4: o algoritmo de Gangal & Ranade é ótimo para árvores de precedência (arborescências).

Prova: Para provar a otimalidade é necessário que se encontre um limite para o escalonamento ótimo e mostrar que o algoritmo de GR atinja esse limite.

Seja J_i o conjunto de vértices no *slot* i.

Seja H a altura da árvore de precedência (quantidade de *slots*).

Será preciso pelo menos $\lceil \frac{J_i}{|P|} \rceil$ passos para escalonar todos os vértices de J_i. Com isso será necessário pelo menos $(i-1)$ *slots* para completar o escalonamento dos vértices restantes da árvores de precedência. A Figura 4.9 é um exemplo típico de árvore de precedência (arborescência), onde denota-se as tarefas atribuídas para ser executada no estágio i por S_i.

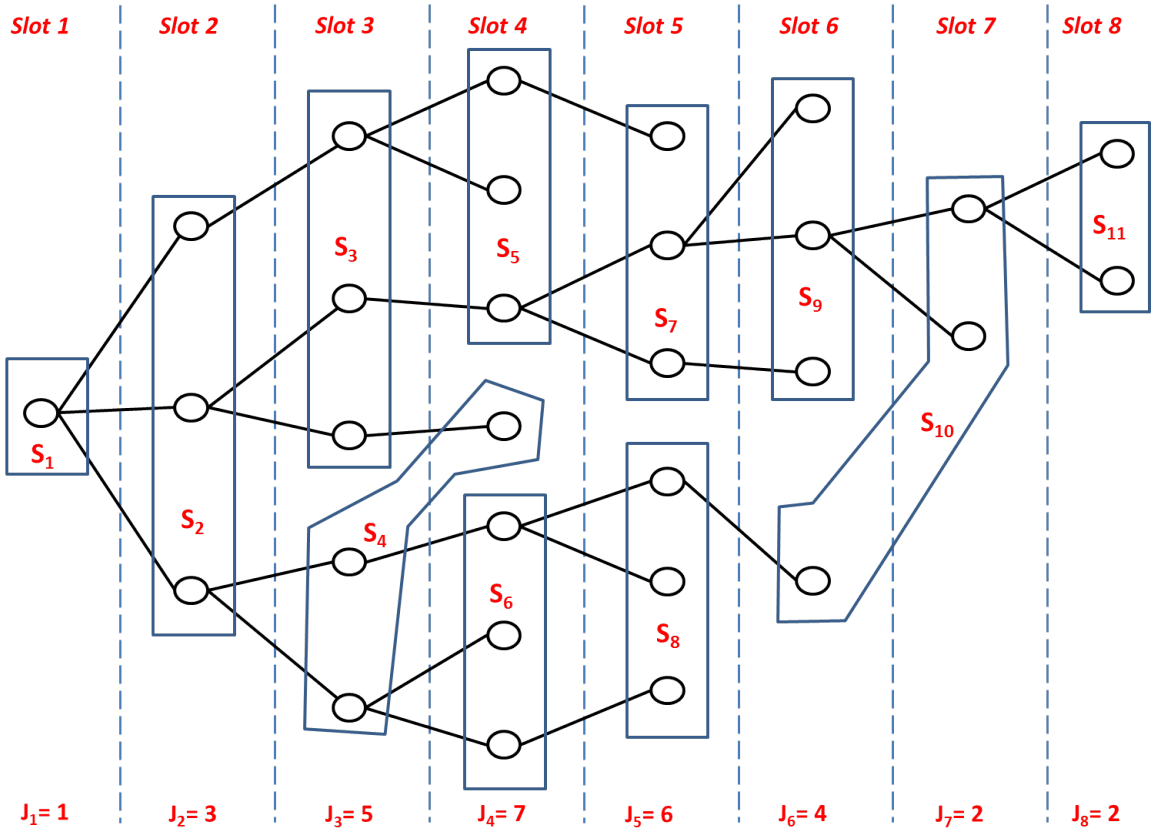

Figura 4.9: Exemplo de uma árvore típica enraizada dividida em *slots* e estágios para $|P| = 3$ para demonstrar limites no escalonamento de GR.

Consequentemente um escalonamento ótimo leva pelo menos

$$\max_{1 \leq i \leq H+1} \{(i-1) + \lceil \frac{J_i}{|P|} \rceil \}. \tag{4.1}$$

Para provar que o escalonamento de GR atinge esse limite será argumentado da seguinte maneira. Seja $S_1, S_2, ..., S_n$ os estágios atribuídos sucessivamente no escalonamento gerado pelo algoritmo de GR.

Seja S_k o último estágio completo no *slot* mais alto (L) em que todos os vértices são desse mesmo *slot*. Se não existir tal estágio, então $S_k = S_0 = \emptyset$. Seja L' a altura dos *slots* mais a esquerda depois de S_k. Observa-se que, pela definição, após S_k se houver algum estágio completo será de multi-*slot*, pois todos os *slots* podem ter menos que $|P|$ vértices.

Após S_k o escalonamento terá exatamente L' para completar todos os passos. Fazendo referência a Figura 4.9 $S_k = S_9$ e $L' = L = 3$. Pela definição de S_k irá existir menos que $|P|$ vértices do mesmo *slot* em L'. Pela definição do limite de latência, mostrada no Capítulo 3, o resto dos vértices será no máximo $1 + slots\ parciais$.

Por definição existe pelo menos $|P|$ vértices alocado até o estágio k no *slot* L. Desde que o algoritmo de GR escalona os de menor prazo primeiro e portanto iniciando pelo *slot* 1, então todos os vértices atribuídos antes de S_k deve ter sido alocado até J_L. Isso prova também que todos os processadores devem ter sido colocados a disposição até o estágio k. Com isso existirá dois casos.

Caso 1: se todos os vértices no *slot* L não estiver em S_k, então $L' = L$ e o tempo de escalonamento será

$$L' + \lceil \frac{|S_1 \cup S_2 \cup ... \cup S_k|}{|P|} \rceil.$$

Por definição, existe menos de $|P|$ vértices no *slot* L depois do estágio k, portanto pode ser escrito

$$(L'-1) + \lceil \frac{|J_{L'}|}{|P|} \rceil.$$

Caso 2: se todos os vértices no *slot* L estiver em S_k, então $L' > L$ e portanto $L' = L-1$ e o tempo de escalonamento será

$$L' + \lceil \frac{|J_{L'} + 1|}{|P|} \rceil.$$

Desde que $\frac{|J_{L'}+1|}{|P|}$ seja inteiro nesse caso, então em ambos os casos o tempo de escalonamento atinge o limite inferior estabelecido anteriormente em 4.1, portanto o escalonamento de GR é ótimo para esse tipo especifico de classe de precedência.

□

4.3 Análise teórica comparativa das razões de aproximação dos algoritmos de Hu, CG e GR

Foi apresentado três algoritmos aproximativos, assim como um algoritmo eficiente e analisado o comportamento de cada um deles juntamente com suas razões de aproximação dos três principais exposto nessa pesquisa com relação ao problema $P|prec, p_j = 1|C_{max}$.

Tabela 4.1: Tabela comparativa da razão de aproximação de cada um dos algoritmos (Hu, CG e GR) para até 30 processadores.

Número de Processadores	**T. C Hu** $2-\frac{1}{\lvert P\rvert-1}$	**CG** $2-\frac{2}{\lvert P\rvert}$	**GR** $2-\frac{7}{3\lvert P\rvert+1}$
2	1	1	1
3	$\frac{3}{2}$	$\frac{4}{3}$	$\frac{13}{10}$
4	$\frac{5}{3}$	$\frac{3}{2}$	$\frac{19}{13}$
5	$\frac{7}{4}$	$\frac{8}{5}$	$\frac{25}{16}$
6	$\frac{4}{3}$	$\frac{5}{3}$	$\frac{31}{19}$
10	$\frac{17}{9}$	$\frac{18}{10}$	$\frac{55}{31}$
15	$\frac{27}{14}$	$\frac{28}{15}$	$\frac{85}{46}$
20	$\frac{37}{19}$	$\frac{38}{20}$	$\frac{115}{61}$
30	$\frac{57}{29}$	$\frac{58}{30}$	$\frac{175}{91}$

Para resultados nessa pesquisa foi reescrito os algoritmos de Hu, Coffman & Graham e de Gangal & Ranade explicando detalhadamente cada algoritmo e seus funcionamentos, assim como fazendo análise de suas razões de aproximação que não foram encontradas

com mais detalhes na literatura.

Para ajudar e facilitar na análise dos algoritmos de Hu e GR foi implementado em linguagem C/C++ ambos algoritmos para análise de desempenho. Foi analisado a possível causa de que o algoritmo de GR tem melhor aproximação para o caso de $|P| \geq 3$ e que nesse trabalho foi explorado também para casos específicos de grafo (*in-tree e out-tree*).

A Tabela 4.1 mostra que é possível ver resultados da comparação entre as aproximações dos três principais algoritmos (Hu, CG e GR) tratado nesse estudo.

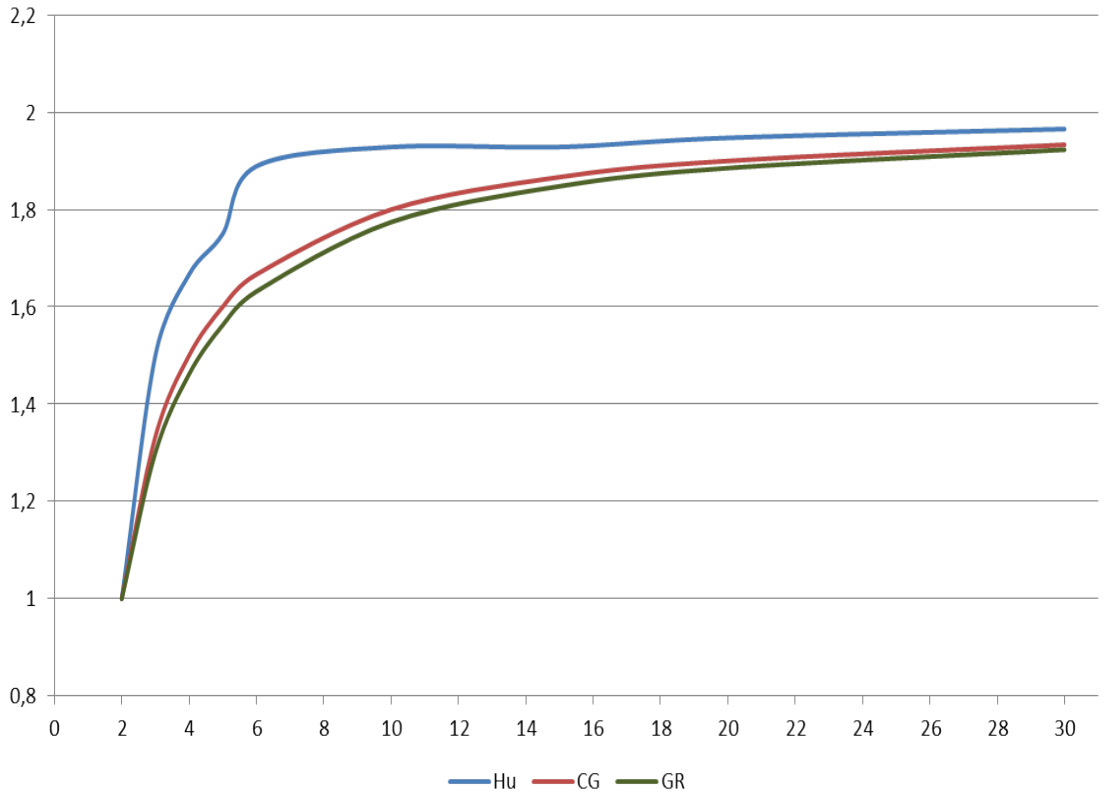

Figura 4.10: Exemplo de gráfico comparativo de cada razão de aproximação dos algoritmos de Hu, CG e GR.

Segundo o que a tabela 4.1 mostra, quando é utilizado dois processadores ambos algoritmos dão solução ótima. Se olhar assintoticamente com o crescimento da quantidade de processadores a diferença entre o algoritmo de CG e GR fica muito pequena, se comparar ambos com o algoritmo de Hu há maior percepção dessa diferença como apresenta a Figura 4.10.

É importante analisar também que Gangal & Ranade (GR) se preocuparam em propor um algoritmo que apresentasse uma razão de aproximação melhor, porém em relação ao tempo de processamento o algoritmo tem maior ordem de complexidade que os anteriores

e isso faz com que seja mais lento computacionalmente que os demais para as mesma classes de instâncias.

A Figura 4.10 apresenta o comportamento assintótico das aproximações de Hu, CG e GR baseado nos dados da Tabela 4.1, onde a curva azul representa a razão de Hu, a vermelha CG e a verde representando GR no eixo das abscissas são os processadores e no eixo das ordenadas as razões. Isso significa que quanto mais próximo de 1 melhor é a aproximação que nesse caso GR se mostra na vantagem. A partir de 15 processadores o crescimento se torna mais lento e CG fica bem próximo de GR demonstrando ser ainda uma boa aproximação. Hu é o pior caso entre os três apresentados.

Observando de maneira geral para o comportamento, tempo de processamento e aproximação do ótimo de cada um dos três algoritmos apresentados na Tabela 4.1 e no gráfico da Figura 4.10 percebe-se nitidamente que o algoritmo de Coffman & Graham (CG) ainda continua sendo bem aceitável por ter boa aproximação e como é um algoritmo que usa estrategia de lista é tido na literatura como algoritmo fácil [Lam and Sethi.(1977)] até mesmo para sua implementação em alguma linguagem.

LISTA DE PRIORIDAD	Razão de Aproximação	|P|=2	|P|≥3					
			In-tree Out-tree	OF	Interval	Quasi-interval	Over-interval	Arbitrary
	Lista	$\frac{3}{2}$						$2-\frac{1}{\|P\|}$
	Hu	opt	opt	opt	opt	opt	opt	$2-\frac{1}{\|P\|-1}$
	MSF	$\frac{4}{3}$			opt	opt		$2-\frac{1}{\|P\|+1}$
	CG	opt	opt		opt	opt	opt	$2-\frac{2}{\|P\|}$
FKN		opt						
GJ		opt						2-aprox
GR		opt	opt	opt	opt	opt	opt	$2-\frac{7}{3\|P\|+1}$

Figura 4.11: Tabela de resultados para DAGs nos algoritmos da literatura com destaque em árvores de precedência por GR.

Em 4.11 é mostrado alguns resultados algorítmicos da literatura em relação as DAGs (*in-tree, out-tree, opposing forest, interval, quasi-interval e over-interval*) sendo destacado (em vermelho) o ótimo pelo algoritmo de Gangal & Ranade (GR) para tais classes

de grafos apresentado nesse trabalho como parte de um dos resultados obtido nessa pesquisa. Alguns dos outros resultados encontram-se na literatura e podem ser achados em [McHugh(1984)].

4.4 Resumo do capítulo

Neste capítulo foram apresentas análises entre comportamento e características de alguns algoritmos aproximativos da literatura, mas que dependendo da classe de grafo de entrada ele pode alcançar o ótimo. Foi provado que para árvore de precedência o algoritmo de Gangal & Ranade (GR) é ótimo.

Foi apresentado também uma análise comparativa entre as razões de aproximação dos três principais algoritmos da literatura apresentados nesta pesquisa (Hu, CG e GR), assim como uma tabela com resultados da literatura incluindo um dos resultados obtidos neste trabalho em relação ao algoritmo de GR e sua otimalidade para árvores de precedências para que se cumpra métodos propostos neste trabalho.

Capítulo 5

Conclusões

Neste trabalho, foi atacado o problema de escalonamento com restrição de precedência e tempo unitário das tarefas em processadores ou máquinas paralelas idênticas variáveis e fixas, este último chamado de "Open 8" se o fixo for o caso de $k = 3$. Verificou-se a influência do número de processadores variáveis $|P|$ ou fixos k sobre a complexidade dos problemas e a razão de aproximação de três algoritmos aproximativos quando $|P| \geq 3$.

Foi mostrado que no caso de $|P| = 2$ os quatro algoritmos pesquisados dão solução ótima quando a instancia for DAGs ou POSets tendo ou não fecho de redução transitiva, contudo para o caso de $|P| \geq 3$ é apresentada razão de aproximação em relação ao ótimo de cada algoritmo estudado e mostrada a prova por dedução da aproximação nos três algoritmos principais deste trabalho, que se encontram no Capítulo 3.

O algoritmo aproximativo mais conhecido pelas restrições gerais de precedência, algoritmo de (CG) (2), foi proposto há mais de 35 anos por Coffman & Graham, mas a sua razão de aproximação foi superada em 2008 por [Gangal and Ranade.(2008)] ambos usam a ideia de grafos que podem ser divididos em subgrafos disjuntos chamado de blocos que é na verdade um conjunto de *slots* e cada *slot* tendo um conjunto de vértices com seus respectivos prazos que é o menor tempo em que cada tarefa pode ser executada quando submetida a um escalonamento na linha do tempo.

Foi explanado alguns algoritmos aproximativos da literatura que resolvem o problema em tempo polinomial provando a razão de aproximação de três deles e mostrando desde os clássicos até os mais atuais por ordem cronológica.

Dentre os algoritmos reescritos da literatura foi feita uma análise comparativa entre Hu e GR para verificação do que foi a melhoria de GR em relação ao Hu. Um dos resul-

tados de maior ênfase deste trabalho foi a prova de otimalidade do algoritmo de GR para determinadas classes de instâncias especificas (árvores de precedência) que tinha como objetivo agregar valor e colaboração na área da otimização em ciência da computação.

5.1 Publicações

Os trabalhos publicados e apresentados como fruto deste estão em ordem cronológica listado a seguir:

- **Trabalho no I-ETC (Encontro de Teoria da Computação) XXXVI-Congresso da Sociedade Brasileira de Computação-CSBC 2016.**
  - Análise de um algoritmo aproximativo para o problema de escalonamento de tarefas com restrições de precedência em máquinas paralelas idênticas [Lever and Latorre(2016)].
- **Trabalho no 3º Encontro Regional de Pesquisa Operacional do Norte (ERPO) e 2º Seminário de otimização Aplicada à Engenharia-2016.**
  - Algoritmo aproximativo para escalonamento de tarefas com restrição de precedência e tempo unitário em processadores idênticos paralelos [Lever and Freitas(2016)].

5.2 Trabalhos futuros

Com base no processo de introdução e análise, pode-se perceber que há muitos caminhos interessantes para futuras pesquisas nesse ramo de escalonamento de tarefas com restrição de precedência.

Investigar outros algoritmos da literatura relacionados ao problema $P|prec, p_j = 1|C_{max}$ pode ser fonte de informação para futuras melhorias. O algoritmo de Gangal & Ranade até o presente momento é o algoritmo com melhor aproximação, contudo sua complexidade computacional em relação aos demais ainda é um fator a se melhorar e isso pode ser feito de maneira a refina-lo conforme Coffman & Graham fizeram com o algoritmo de Hu.

O "Open 8" (escalonamento com restrição de precedência para três processadores) como ficou conhecido é uma área muito teórica, porém promissora para investigações no qual o objetivo é encontrar uma nova classe de ordens que podem ser resolvidos por meio de algoritmos conhecidos ou suas generalizações. Outro desafio para a pesquisa nessa área é encontrar algoritmos com melhor razão de aproximação, com tempo de complexidade computacional reduzido e desempenho eficiente para o problema $P|prec, p_j = 1|C_{max}$.

Investigar casos especiais do problema com número fixo de processadores, que continua em aberta sua complexidade pode ser considerado muito promissor, pois o problema está a algumas décadas sendo estudado na ciência da computação e ainda continua em aberto até hoje, isso pode despertar o interesse em pesquisadores a ir em busca desse conhecimento e quem sabe provar onde está inserida sua classe de complexidade.

Referências

[Braschi and Trystram(1994)] Braschi, B., Trystram, D., 1994. A new insight into the coffman-graham algorithm. SIAM J. Comput. 23, 662–669.

[Chen(1975)] Chen, N. F., 1975. An analysis of scheduling - algorithms in multiprocessing computing systems. Technical report UIUCDCS-R-75-724, Department of computer science Univesity of Illinois at urbana-champaign.

[Chen and Liu(1975)] Chen, N. F., Liu, C. L., 1975. On a class of scheduling algorithms for multiprocessors computing systems. Personal commnication, December, 1–16.

[Coffman Jr and Graham(1972)] Coffman Jr, A. P. E., Graham, R. L., 1972. Optimal scheduling for two-processor systems. Acta informatica 1 (3), 200–213.

[Coffman Jr. and Garey(1993)] Coffman Jr., E., Garey, M., 1993. Proof of the 4/3 conjecture for preemptive vs. nonpreemptive two-processor scheduling. Journal of the ACM 40 (5), 991–1018.

[Cormen et al.(2009)Cormen, Leiserson, Rivest, and Stein] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., 2009. Introduction to Algorithms, Third Edition, 3rd Edition. The MIT Press.

[Gangal and Ranade.(2008)] Gangal, D., Ranade., A., 2008. Precedence constrained scheduling in $(2 - \frac{7}{3p+1})$ para $p \geq 4$ optimal. Elsevier. Journal of Computer and System Sciences, 74 (2008) 1139-1146. Vol:1, 1–13.

[Garey and Johnson(1976)] Garey, M., Johnson, D., 1976. Scheduling tasks with nonuniform deadlines on two processors. J Assoc Comput Mach 23 (3), 461–467.

[Garey and Johnson(1979)] Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman.

[Graham et al.(1979)Graham, Lawler, Lenstra, and Kan] Graham, R. L., Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics.

[Hu(1961)] Hu, T. C., 1961. Parallel sequencing and assembly line problems. IBM Research Center, Yorktotim, New York 1 (1), 8.

[J. Du(1991)] J. Du, J.Y.T. Leung, G. Y., 1991. Scheduling chain-structured tasks to minimize makespan and mean flow time. Information and Computation 2 (1991) 219-236.

[J. K. Lenstra(1978)] J. K. Lenstra, A. H. G. R. K., 1978. Complexity of scheduling under precedence constraints. Operations Research 26 (1), 22–35.

[Lam and Sethi.(1977)] Lam, S., Sethi., R., 1977. Worst case analysis of two scheduling algorithms. SIAM J. COMPUT. N 2 Vol:6, 518–536.

[Lever and Freitas(2016)] Lever, E., Freitas, R., 2016. Algoritmo aproximativo para escalonamento de tarefas com restrição de precedência e tempo unitário em processadores idênticos paralelos. 3º Encontro Regional de Pesquisa Operacional do Norte and 2º Seminário de otimização Aplicada à Engenharia-2016.

[Lever and Latorre(2016)] Lever, E. Freitas, R., Latorre, O., 2016. Análise de um algoritmo aproximativo para o problema de escalonamento de tarefas com restrições de precedência em máquinas paralelas idênticas. I- Encontro de Teoria da Computação-ETC, XXXVI-Congresso da Sociedade Brasileira de Computação-CSBC 2016.

[McHugh(1984)] McHugh, J. A. M., 1984. Hu's precedence tree scheduling algorithm: A simple proof. Naval Research Logistics Quarterly 31 (3), 409–411.

[Papadimitriou and Steiglitz(1998)] Papadimitriou, C., Steiglitz, K., 1998. Combinatorial Optimization: algorithms and complexity. Dover Publications.

[Papadimitriou and Yannakakis(1979)] Papadimitriou, C., Yannakakis, M., 1979. Scheduling interval-ordered tasks. SIAN J. Comput., 8(3): 405-409.

[Ullman(1975)] Ullman, J., 1975. Np-complete scheduling problems. Journal of Computer and System Sciences 10 (3), 384 – 393.

[Zang(2005)] Zang, N., 2005. Uet multiprocessor scheduling problems. University of California, San Diego. Vol:1, 1–9.